SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

January 2018, Vol. 61 012103:1–012103:10
doi: 10.1007/s11432-016-9062-x

Decidability of linearizabilities for relaxed data structures

Chao WANG[1,2]*, Yi LV[1,2] & Peng WU[1,2]

[1] *State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing* 100190*, China;*
[2] *University of Chinese Academy of Sciences, Beijing* 100049*, China*

Received 21 November 2016/Revised 6 February 2017/Accepted 31 March 2017/Published online 1 September 2017

Abstract Many recent implementations of concurrent data structures relaxed their linearizability requirements for better performance and scalability. Quasi-linearizability, k-linearizability and regular-relaxed linearizability are three quantitative relaxation variants of linearizability that have been proposed as correctness conditions of relaxed data structures, yet preserving the intuition of linearizability. Quasi-linearizability has been proved undecidable. In this paper, we first show that k-linearizability is undecidable for a bounded number of processes, by reducing quasi-linearizability into it. We then show that regular-relaxed linearizability is decidable for a bounded number of processes. We also find that the number of the states of a relaxed specification is exponential to the number of the states of the underlying specification automaton (representing its relaxation strategy), and polynomial to the number of the states of the underlying quantitative sequential specification and the number of operations.

Keywords concurrent data structures, quantitative relaxation, linearizability, decidability, finite automata

Citation Wang C, Lv Y, Wu P. Decidability of linearizabilities for relaxed data structures. Sci China Inf Sci, 2018, 61(1): 012103, doi: 10.1007/s11432-016-9062-x

1 Introduction

Developing concurrent data structures often requires subtle synchronization mechanisms, e.g., non-blocking or fine-grained synchronization, to support for concurrency. Recent implementations even relaxed their correctness requirements for better performance and scalability [1–4]. Such relaxations often make relaxed data structures error-prone and notoriously hard to verify.

Linearizability [5] is a well established correctness requirement of concurrent data structures, while the correctness requirements of relaxed data structures have been far less studied. Quasi-linearizability [6] and k-linearizability [7] are two relaxed variants of linearizability. They both relax sequential specifications in a quantitative way, and relate concurrent data structures and their relaxed specifications with the standard notion of linearizability. Recently, we proposed the notion of regular-relaxed linearizability [8], i.e., linearizability with regular-relaxed specifications. Although it is essentially subsumed by k-linearizability, it is expressive enough to cover many relaxed data structures in practice [1–4] and relaxed data structure examples in [6, 7]. More interestingly, regular-relaxed linearizability establishes a balance between the expressiveness of k-linearizability and the complexity in verifying the correctness of relaxed data structures.

* Corresponding author (email: wangch@ios.ac.cn)

The problem of whether a concurrent data structure is linearizable with respect to its regular sequential specification is decidable for a bounded number of processes [9], but undecidable for a unbounded number of processes [10]. Since linearizability can be regarded as a special case of quasi-linearizability, k-linearizability and regular-relaxed linearizability, it is obvious that quasi-linearizability, k-linearizability and regular-relaxed linearizability are also undecidable for an unbounded number of processes. In our previous work [11], we have proved that quasi-linearizability is also undecidable for a bounded number of processes. But the decidability problems of k-linearizablity and regular-relaxed linearizability remain open for a bounded number of processes. In this regard, this paper has the following theoretical contributions.

(1) We prove that k-linearizability is undecidable for a bounded number of processes, i.e., the problem of whether a concurrent data structure is k-linearizable with respect to a regular sequential specification is undecidable for a bounded number of process.

(2) We prove that regular-relaxed linearizability is decidable for a bounded number of processes, i.e., for a relaxed specification built upon a regular quantitative sequential specification, the problem of whether a concurrent data structure is regular-relaxed linearizable with respect to the relaxed specification is decidable for a bounded number of processes.

We show that a relaxed specification of quasi-linearizability is also a relaxed specification of k-linearizability. Therefore, the undecidability result of k-linearizability follows directly from that of quasi-linearizability. Note that to generate a relaxed specification of k-linearizability, we need to first identify a sequential specification as a labelled transition system (LTS), which is then completed and assigned with transition costs. The resulting complete LTS is termed a quantitative sequential specification. Then, we need to determine the costs of the traces of the resulting LTS by using a path cost function, and choose all the traces that have cost no greater than k. Such cost functions characterize the relaxation strategies of k-linearizability. In our proof, the quantitative sequential specification is obtained by identifying the empty sequential specification as an LTS that has just one state with no transitions, and then completing the LTS with self-loops, each labelled with an arbitrary operation and assigned with the same cost. Then, we define a special filtering function that essentially chooses the traces of the resulting LTS that are in the relaxed specification of quasi-linearizability. This proof indicates that by using a specific path cost function, a relaxation process of k-linearizability may transform a regular sequential specification into a relaxed specification that can be far more complex.

Regular-relaxed linearizability uses relaxation strategies that are simple enough to be represented as finite automata (termed specification automata). A relaxed specification of regular-relaxed linearizability is obtained by filtering a quantitative sequential specification with respect to a specification automaton. To obtain the decidability result of regular-relaxed linearizability, we prove that a relaxation process of regular-relaxed linearizability can only result in a regular relaxed specification from a regular quantitative sequential specification. Then, this decidability result follows directly from [9]. This constitutes the first positive decidability result for relaxed variants of linearizability. Hence, regular-relaxed linearizability appears more suitable for verification in practice.

As for its complexity, we further show that a relaxed specification of regular-relaxed linearizability can be represented by a deterministic finite automaton, of which the number of the states is exponential to the number of the states of the underlying specification automaton and polynomial to the number of the states of the underlying quantitative sequential specification and the number of operations. In [8], we constructed specification automata for typical relaxed queues and stacks. These specification automata are in a specific form. In this paper we show that for these specification automata, the corresponding relaxed specifications can be represented by deterministic finite automata, of which the number of the states is polynomial to the number of the states of the underlying quantitative sequential specification and specification automaton, and to the number of operations.

Related work. Quasi-linearizability [6] was the first relaxation scheme for sequential specifications of concurrent data structures. k-linearizabilty [7] may arguably cover more relaxed data structures than quasi-linearizability, with more accurate relaxation processes. In our previous work [8] we proposed regular-relaxed linearizabiltiy. As a preliminary attempt, verification tools have been adapted to model checking quantitatively relaxed data structures [12, 13]. There are already several known results published on the decidability of linearizability and its relaxed variants [9–11, 14, 15]. Specially in a preceding work [11], we proved that quasi-linearizability is undecidable for a bounded number of processes.

2 Notations and terminologies

In this section we introduce the basic notations and terminologies used throughout the paper, including concurrent systems, their operational semantics and linearizability. We then introduce the definition and decidability of quasi-linearizability.

For sequences l and l', let $l \cdot l'$ denote their concatenation. Let $l \uparrow_{\Sigma}$ denote the projection of l onto Σ. Let $\_$ denote an item, of which the value is irrelevant, and ϵ denote the empty sequence. A labelled transition system (LTS) is a tuple $\mathcal{A} = (Q, \Sigma, \rightarrow, q_0)$, where Q is a set of states, Σ is an alphabet of transition labels, $\rightarrow \subseteq Q \times \Sigma \times Q$ is a transition relation and q_0 is the initial state. A path of $\mathcal{A}$ is a finite transition sequence $q_0 \xrightarrow{\alpha_1} q_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_k} q_k$ with $k \geqslant 0$. A finite sequence $t = \alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_k$ with $k \geqslant 0$ is a trace of $\mathcal{A}$ if there exists a path $q_0 \xrightarrow{\alpha_1} q_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_k} q_k$ of $\mathcal{A}$.

2.1 Concurrent system and operational semantics

A concurrent data structure encapsulates a collection of methods for accessing a possibly shared instance of the concurrent data structure. A most general client program is a program that interacts with a concurrent data structure, and aims to exhibit all the possible behaviors of the concurrent data structure. Similar definitions of concurrent data structures and most general client programs can be found in [16].

The operational semantics of a concurrent data structure is typically defined in the context of a concurrent system. A concurrent system for a concurrent data structure consists of a bounded number of processes, each running the most general client program of the concurrent data structure on a separate processor.

Given a concurrent data structure $\mathcal{L} = (\mathcal{X}_{\mathcal{L}}, \mathcal{M}_{\mathcal{L}}, \mathcal{D}_{\mathcal{L}}, Q_{\mathcal{L}}, \rightarrow_{\mathcal{L}})$ and a positive integer n as the number of processes, each process of a concurrent system acts according to the most general client program $(\mathcal{M}_{\mathcal{L}},\mathcal{D}_{\mathcal{L}},\{q_c\},\rightarrow_{\mathrm{mgc}})$. Then, the operational semantics is defined as an LTS $[\![\mathcal{L}, n]\!]_{\mathrm{cs}} = (\mathrm{Conf}_{\mathrm{cs}}, \Sigma_{\mathrm{cs}}, \rightarrow_{\mathrm{cs}}, \mathrm{InitConf}_{\mathrm{cs}})$, where $\mathrm{Conf}_{\mathrm{cs}}, \Sigma_{\mathrm{cs}}, \rightarrow_{\mathrm{cs}}, \mathrm{InitConf}_{\mathrm{cs}}$ are defined as follows.

Each configuration of $\mathrm{Conf}_{\mathrm{cs}}$ is a tuple (p, d), where
– $p : \{1, \ldots, n\} \rightarrow \{q_c\} \cup Q_{\mathcal{L}}$ represents control states of each process;
– $d : \mathcal{X}_{\mathcal{L}} \rightarrow \mathcal{D}_{\mathcal{L}}$ represents values at each memory location;
– $\mathrm{InitConf}_{\mathrm{cs}} = (p_{\mathrm{init}}, \mathrm{InitV}_{\mathcal{L}})$ is the initial configuration in $\mathrm{Conf}_{\mathrm{cs}}$, where $p_{\mathrm{init}}(i) = q_c$ for each process i and $\mathrm{Init}V_{\mathcal{L}} : \mathcal{X}_{\mathcal{L}} \rightarrow \mathcal{D}_{\mathcal{L}}$ is an initial valuation for memory locations.

Σ_{cs} is the set of transition actions including internal actions $\tau(i)$, read actions $\mathrm{read}(i, x, a)$, write actions $\mathrm{write}(i, x, a)$, cas actions $\mathrm{cas}(i, x, a, b)$, call actions $\mathrm{call}(i, m, a)$, return actions $\mathrm{return}(i, m, a)$ for $1 \leqslant i \leqslant n, x \in \mathcal{X}_{\mathcal{L}}, a, b \in \mathcal{D}_{\mathcal{L}}$ and $m \in \mathcal{M}_{\mathcal{L}}$.

$\rightarrow_{\mathrm{cs}} \in \mathrm{Conf}_{\mathrm{cs}} \times \Sigma_{\mathrm{cs}} \times \mathrm{Conf}_{\mathrm{cs}}$ is the least transition relation satisfying the transition rules shown in Figure 1.

- Tau rule: A τ command only influences the control state of the current process.

- Read and Write rules: A read action takes the value of a memory location in memory, and a write action changes the value of a memory location in memory directly.

- Cas-Success and Cas-Fail rules: A successful cas command changes the value of a memory location in memory immediately, while a failed cas command fails to do so.

- Call and Return rules: Given a $\mathrm{call}(i, m, a)$ command, the current process starts to execute the initial state $\mathrm{is}_{(m,a)}$ of method m with value a. When the process comes to the final state $\mathrm{fs}_{(m,a)}$ of method m with return value a, it launches a return action and starts to execute the most general client program again.

$$\frac{p(i) = q_1, q_1 \xrightarrow{\tau}_{\mathcal{L}} q_2}{(p,d) \xrightarrow{\tau(i)}_{\mathrm{cs}} (p[i:q_2], d)}\text{Tau}$$

$$\frac{p(i) = q_1, q_1 \xrightarrow{\mathrm{read}(x,a)}_{\mathcal{L}} q_2, d(x) = a}{(p,d) \xrightarrow{\mathrm{read}(i,x,a)}_{\mathrm{cs}} (p[i:q_2], d)}\text{Read}$$

$$\frac{p(i) = q_1, q_1 \xrightarrow{\mathrm{write}(x,a)}_{\mathcal{L}} q_2}{(p,d) \xrightarrow{\mathrm{write}(i,x,a)}_{\mathrm{cs}} (p[i:q_2], d[x:a])}\text{Write}$$

$$\frac{p(i) = q_1, q_1 \xrightarrow{\mathrm{cas\_suc}(x,a,b)}_{\mathcal{L}} q_2, d(x) = a}{(p,d) \xrightarrow{\mathrm{cas}(i,x,a,b)}_{\mathrm{cs}} (p[i:q_2], d[x:b])}\text{Cas-Success}$$

$$\frac{p(i) = q_1, q_1 \xrightarrow{\mathrm{cas\_fail}(x,a,b)}_{\mathcal{L}} q_2, d(x) \neq a}{(p,d) \xrightarrow{\mathrm{cas}(i,x,a,b)}_{\mathrm{cs}} (p[i:q_2], d)}\text{Cas-Fail}$$

$$\frac{p(i) = q_c, m \in \mathcal{M}_{\mathcal{L}}, a \in \mathcal{D}_{\mathcal{L}}}{(p,d) \xrightarrow{\mathrm{call}(i,m,a)}_{\mathrm{cs}} (p[i:is_{(m,a)}], d)}\text{Call}$$

$$\frac{p(i) = \mathrm{fs}_{(m,a)}}{(p,d) \xrightarrow{\mathrm{return}(i,m,a)}_{\mathrm{cs}} (p[i:q_c], d)}\text{Return}$$

Figure 1 Transition rules of $\rightarrow_{\mathrm{cs}}$.

2.2 Linearizability

Linearizability [5] is the de facto standard correctness condition for concurrent data structures. An execution of the concurrent system is modeled by a history, which is a finite sequence of call and return actions. A call action call(i, m, a) happens when a method m is called by process i with argument a. A return action happens when a called method m of process i returns a to the calling process.

Intuitively, a history is linearizable, if every method of the history takes effect instantaneously at some point between the call and the return action of the method. Let us formally introduce the definition of linearizability. A return action return(i_1, m_1, a_1) matches a call action call(i_2, m_2, a_2), if $i_1 = i_2 \wedge m_1 = m_2$. Then, a history is sequential if it starts with a call action and each call (respectively, return) action is immediately followed by a matching return (respectively, call) action unless it is the last action. A process sub-history $h|_i$ is the history consisting of all and only the actions of process i of history h. Then, two histories h_1 and h_2 are equivalent, if $h_1|_i = h_2|_i$ for each process i. Let complete(h) be the maximal subsequence of a history h consisting of all the matching call and return actions. An operation $m(a, b)$ represents that method m is called with argument a and returns value b. We say an operation $e = m(a, b)$ is in a history, if the history contains a pair consisting of a call action, inv(e) = call(i, m, a), and the next matching return action, res(e) = return(i, m, b) for some process i.

An operation $m(a, b)$ represents that method m is called with argument a and returns value b. A (sequential) specification of a concurrent data structure is defined as a prefix-closed set of sequences over $\{m(a, b) | m \in \mathcal{M}, a, b \in \mathcal{D}\}$. A history h is linearizable with respect to a sequential specification Spec, if h can be extended to a history h_0 by appending zero or more return actions, where there exists sequence $m_1(a_1, b_1) \cdot \ldots \cdot m_u(a_u, b_u) \in$ Spec and processes $i_1, \ldots, i_u$, such that

- complete(h_0) is equivalent to the sequential history $h' =$ call$(i_1, m_1, a_1) \cdot$ return$(i_1, m_1, b_1) \cdot \ldots \cdot$ call$(i_u, m_u, a_u) \cdot$ return(i_u, m_u, b_u).
- For any operations e_1, e_2 of h, if res(e_1) precedes inv(e_2) in h, then this also holds in h'.

Let history$([\![\mathcal{L}, n]\!]_{\mathrm{cs}})$ be the set of histories of $[\![\mathcal{L}, n]\!]_{\mathrm{cs}}$. A concurrent data structure $\mathcal{L}$ is linearizable with respect to a sequential specification Spec for n processes, if each history in history$([\![\mathcal{L}, n]\!]_{\mathrm{cs}})$ is. Our definition of sequential specification and linearizability complies with the definitions in [6,7] and essential respect the definitions in [5].

Since $[\![\mathcal{L}, n]\!]_{\mathrm{cs}}$ is finite-state, it is easy to see that history($[\![\mathcal{L}, n]\!]_{\mathrm{cs}}$) is a regular set. Alur et al. [9] have proved that it is decidable whether a regular set of histories is linearizable with respect to a regular sequential specification. Therefore, it is decidable whether a concurrent data structure is linearizable with respect to a regular sequential specification, as shown by the following lemma.

Lemma 1 (Decidability of linearizability [9]). Given a concurrent data structure $\mathcal{L}$ and a regular sequential specification Spec, the problem of whether $\mathcal{L}$ is linearizable with respect to Spec for n processes is decidable.

2.3 Definition and decidability of quasi-linearizability

Quasi-linearizability [6] is a quantitatively relaxed variant of linearizability. It relaxes the sequential specification in a quantitative way and relates the concurrent data structure and relaxed specifications with the standard notion of linearizability. The relaxation of quasi-linearizability is carried out in a "syntactic" way [7] and is based on permutation.

Let us show the relaxation process of quasi-linearizability to sequential specifications. For each element α in a sequence l, let $l[\alpha]$ denote its index. For two sequences l_1, l_2 of operations, distance$(l_1, l_2) = \max\{|l[\alpha] - l[\alpha]| \ |\alpha \text{ is in } l\}$ is the distance between l_1 and l_2. The relaxation of quasi-linearizability is guided by a function $Q : P \to \mathbb{N}$ called quasi-linearization factor, where P is a set containing subsets of operations. The relaxed specifications Q-Spec(Spec) of quasi-linearizability, called Q-quasi-sequential specification, is obtained from sequential specification Spec and quasi-linearization factor Q. A sequence s is in Q-Spec(Spec), if it is a prefix of some sequence s', and there exists sequence $s'' \in$ Spec, such that for each d in domain of Q, distance$(s'|_d, s''|_d) \leqslant Q(d)$. A history h is Q-quasi-linearizable with respect to sequential specification Spec, if h is linearizable with respect to Q-Spec(Spec). A concurrent data structure L is Q-quasi-linearizable with respect to sequential specification Spec, if each history in history($[\![\mathcal{L}, n]\!]_{\mathrm{cs}}$) is.

In a preceding work [11] we have proved that it is undecidable whether a concurrent data structure is quasi-linearizable with respect to a regular sequential specification for a finite number of processes (even so for only one process), as shown by the following lemma.

Lemma 2 (Undecidability of quasi-linearizability [11]). Given a concurrent data structure $\mathcal{L}$, a quasi-linearizabilization factor Q and a regular sequential specification Spec, the problem of whether $\mathcal{L}$ is Q-quasi-linearizable with respect to Spec for n processes is undecidable.

3 Definitions of k-linearizability and regular-relaxed linearizability

In this section we introduce the definitions of k-linearizability and regular-relaxed linearizability. Similar to quasi-linearizability, they are also quantitatively relaxed variants of linearizability. However, they relax sequential specification in a "semantical" way [7], where the sequential specification should be identified as an LTS and then the cost of each transition of the LTS can be determined.

3.1 Definition of k-linearizability

k-linearizability [7] is a quantitatively relaxed variant of linearizability. It relaxes the sequential specification in a quantitative way and relates the concurrent data structure and relaxed specifications with the standard notion of linearizability.

Roughly speaking, the relaxation process contains three steps. First, the sequential specification is identified with a particular LTS whose states are sets of sequences with indistinguishable future behavior. Then, we specify costs of operations and traces of this LTS. Finally, the relaxed specification is the set of traces with certain costs. An operation has cost r, if it has distance r from a corresponding "normal" operation, such as dequeuing an element from a queue that is r elements away from the head of the queue. A trace has cost r, if it has distance r from a corresponding "normal" operation sequence, such as a trace where each of its operation has cost less or equal than r.

Given a sequential specification Spec, the detailed relaxation process of k-linearizability in [7] for Spec is shown as follows. First, we need to identify Spec with a particular LTS LTS(Spec). A relation $\equiv_{\rm Spec}$ is constructed as follows: Given two sequences ${\rm seq}_1, {\rm seq}_2$ of operations, ${\rm seq}_1 \equiv_{\rm Spec} {\rm seq}_2$, if for each sequence ${\rm seq}_3$ of operations, ${\rm seq}_1 \cdot {\rm seq}_3 \in$ Spec if and only if ${\rm seq}_2 \cdot {\rm seq}_3 \in$ Spec. It is obvious that $\equiv_{\rm Spec}$ is an equivalence relation, and let $[{\rm seq}]_{\rm Spec}$ be the equivalent class of seq with respect to $\equiv_{\rm Spec}$. An LTS ${\rm LTS(Spec)} = (Q, \Sigma, \rightarrow, q_0)$ is constructed to identify Spec, where (1) each state is an equivalent class of $\equiv_{\rm Spec}$, (2) each transition label is an operation, (3) $\rightarrow = \{[t]_{\rm Spec} \xrightarrow{m(a,b)} [t \cdot m(a,b)]_{\rm Spec} \mid t \cdot m(a,b) \in {\rm Spec}\}$ is the transition relation and (4) $q_0 = [\epsilon]$ is the initial state.

Then, we construct the LTS ${\rm QLTS(Spec)} = (Q, \Sigma, Q \times \Sigma \times Q, q_0, C, f)$, which makes transitions between all the pairs of the states of LTS(Spec) with all the operations as labels, and use a transition cost function $f \colon Q \times \Sigma \times Q \rightarrow C$ to determine the cost of each transition. Herein, C is a well-ordered cost domain with 0 the minimum, and f maps only the transitions of LTS(Spec) to 0. QLTS(Spec) is called a quantitative sequential specification.

A quantitative sequence is a sequence $(o_1, r_1) \cdot \ldots \cdot (c_l, r_l)$, where o_i is an operation and $r_i \in C$ is its cost. Let qtr(Spec) denote the set of the quantitative sequences of Spec in QLTS(Spec), i.e., ${\rm qtr(Spec)} = \{(o, r_1) \cdot \ldots \cdot (o_l, r_l) | o_1 \cdot \ldots \cdot o_l \in {\rm Spec}, (o_1, r_1) \cdot \ldots \cdot (c_l, r_l)$ is a trace of QLTS(Spec)$\}$. A path cost function $g : {\rm qtr(Spec)} \rightarrow C$ is then used to determine the cost of each quantitative sequence. Herein, g is required to satisfy that, for any sequences s_1 and s_2, if s_1 is a prefix of s_2, then $g(s_1) \leqslant g(s_2)$.

At last, a relaxed specification ${\rm Spec}_k$ of k-linearizability, called k-relaxed specification, is obtained by choosing the sequences with costs no greater than k for $k \in C$. Given sequence $s = o_1 \cdot \ldots \cdot o_l \in {\rm Spec}$, let ${\rm qtr}(s) = \{(o_1, r_1) \cdot \ldots \cdot (o_l, r_l) \in {\rm qtr(Spec)}\}$ represent the set of all the quantitative sequences for s. Then, an operation sequence s is in the k-relaxed specification ${\rm Spec}_k$, if $\min\{g(\tau) | \tau \in {\rm qtr}(s)\} \leqslant k$, or, in other words, some quantitative sequence of s has cost no greater than k.

3.2 Definition of regular-relaxed linearizability

Recently, we propose a quantitative relaxed variant of linearizability called Linearizability with regular-relaxed specification [8], or shortened as regular-relaxed linearizability. It is also a quantitatively relaxed variant of linearizability. It relaxes the sequential specification in a quantitative way and relates the concurrent data structure and relaxed specifications with the standard notion of linearizability. It is essentially a subset of k-linearizability, and it intends to balance the expressiveness of k-linearizability with the complexity in verifying the correctness of relaxed data structures.

Regular-relaxed linearizability uses finite automata (called specification automata) to represent relaxation strategies. We identify a set of input methods that takes one input argument, such as the typical enq method of queue and push method of stack. Let $\mathcal{M}_{\rm inp} \subseteq \mathcal{M}$ be the set of input methods in $\mathcal{M}$ and $M_{\rm oth} = \mathcal{M} - \mathcal{M}_{\rm inp}$. Given $k \in \mathbb{N}$, let Nat, $E(k)$, $N(k)$, LE(k) and $G(k)$ represent predicate "is a natural number", "equal to k", "not equal to k", "less than or equal to k" and "greater than k", respectively. Let $M^{p}_{\rm inp}$, $M^{p\text{-}v}_{\rm oth}$ and $M^{p\text{-}n}_{\rm oth}$ represent the predicate that identifies $\mathcal{M}_{\rm inp}$ operations, $\mathcal{M}_{\rm oth}$ operations that return values and $\mathcal{M}_{\rm oth}$ operations that return null, respectively. Then, a specification automaton is a finite automaton $\mathcal{A} = (Q_{\mathcal{A}}, F_{\mathcal{A}}, \Sigma_{\mathcal{A}}, \rightarrow_{\mathcal{A}}, q_{\rm init})$, where

- $Q_{\mathcal{A}}$ is the finite set of states; $q_{\rm init} \in Q_{\mathcal{A}}$ is the initial state, and $F_{\mathcal{A}} = Q_{\mathcal{A}}$ is the set of final states.
- $\Sigma_{\mathcal{A}}$ is a finite set of transition labels, each of which is represented as a pair (po, pc) of predicates. po $\in \{M^{p}_{\rm inp}, M^{p\text{-}v}_{\rm oth}, M^{p\text{-}n}_{\rm oth}\}$ is used to select certain operations; while pc $\in \{{\rm Nat}, N(0), E(0)\} \cup \{{\rm LE}(i), {\rm LE}(i) \wedge N(0), G(i) | i \in \mathbb{N}\}$ is used to select certain costs.
- $\rightarrow_{\mathcal{A}} \subseteq Q_{\mathcal{A}} \times \Sigma_{\mathcal{A}} \times Q_{\mathcal{A}}$ is the transition relation.

Roughly speaking, the relaxation of regular-relaxed linearizability contains two steps. First, we identified sequential specification as an LTS and determine costs of each operations of the LTS, resulting in a quantitative sequential specification. Then, we generate the relaxed specification by filtering the LTS with the specification automaton. Given a sequential specification Spec, the detailed relaxation process of regular-relaxed linearizability in [8] for Spec is shown below.

First, we identify Spec with a particular LTS $\mathrm{LTS}_r(\mathrm{Spec}) = (Q, \Sigma, \rightarrow, q_0)$, and the only requirement to $\mathrm{LTS}_r(\mathrm{Spec})$ is that the set of its traces must be Spec. Let ∞ be a special number that is greater than any natural number. Similarly as [7], we generate a quantitative sequential specification $\mathrm{QLTS}_r(\mathrm{Spec}) = (Q, \Sigma, Q \times \Sigma \times Q, q_0, \mathbb{N} \cup \{\infty\}, f)$ from $\mathrm{LTS}_r(\mathrm{Spec})$. Here f determines cost of operations, and the domain of cost is fixed to $\mathbb{N} \cup \{\infty\}$.

Then, the relaxed specifications $\mathrm{Spec}_{\mathcal{A}}$ of regular-relaxed linearizability is obtained by filtering the traces of $\mathrm{QLTS}_r(\mathrm{Spec})$ with specification automaton $\mathcal{A}$: an operation sequence $s = o_1 \cdot \ldots \cdot o_l$ is in $\mathrm{Spec}_{\mathcal{A}}$, if there exist costs $r_1, \ldots, r_l \in \mathbb{N}$, states $q_1, \ldots, q_{l+1}$ and transitions $q_1 \xrightarrow{(\mathrm{po}_1, \mathrm{pc}_1)}_{\mathcal{A}} q_2, \ldots, q_l \xrightarrow{(\mathrm{po}_l, \mathrm{pc}_l)}_{\mathcal{A}} q_{l+1}$ of $\mathcal{A}$, such that s is a trace of $\mathrm{QLTS}_r(\mathrm{Spec})$, $q_1 \equiv q_{\mathrm{init}}$, and for each $1 \leqslant i \leqslant l$, o_i satisfies po_i, r_i satisfies pc_i.

4 Undecidability of k-linearizability

Essentially, quasi-linearizability uses quasi-linearization factors to characterize its relaxation strategies (which relax sequential specifications through permutation), while k-linearizability semantically identifies a sequential specification as an LTS and uses path cost functions to characterize its relaxation strategies. In this section we prove that a relaxed specification of quasi-linearizability is also a relaxed specification of k-linearizability. Then, as shown below, the undecidability result of k-linearizability follows directly from that of quasi-linearizability in Lemma 2. This result implies that k-linearizability is more expressive than quasi-linearizability in their relaxation capabilities.

Let $\mathrm{Op}(\mathcal{M}, \mathcal{D}) = \{m(a, b) | m \in \mathcal{M}, a, b \in \mathcal{D}\}$ be the set of operations with method names in $\mathcal{M}$ and data domain $\mathcal{D}$.

Theorem 1 (Undecidability of k-linearizability)**.** Given a concurrent data structure $\mathcal{L}$, a regular sequential specification Spec and an element k of a well-ordered cost domain, the problem of whether $\mathcal{L}$ is k-linearizable with respect to Spec for n processes is undecidable.

Proof. Given sequential specification $\mathrm{Spec} \subseteq \mathrm{Op}(\mathcal{M}, \mathcal{D})^*$ and a quasi-linearization factor Q, let us show how to use the relaxation process of k-linearizability to generate Q-Spec(Spec).

Let sequential specification $\mathrm{Spec}_{\mathrm{emp}} = \emptyset$. The LTS for identifying $\mathrm{Spec}_{\mathrm{emp}}$ is $(\{q_0\}, \mathrm{Op}(\mathcal{M}, \mathcal{D}), \emptyset, q_0)$, which contains only one state $q_0 = [\epsilon]$ and no transition. Then, we construction quantitative sequential specification $\mathrm{QLTS}(\mathrm{Spec}_{\mathrm{emp}})$ as follows:

- We construct a well-ordered cost domain $C' = \mathrm{Op}(\mathcal{M}, \mathcal{D}) \cup \{0, 1\}$. Since $\mathcal{M}$ and $\mathcal{D}$ are both finite, there must be total order $<_{\mathcal{M}}$ and $<_{\mathcal{D}}$ for them, respectively. A order $<_{C'}$ for elements in C' is defined as follows: (1) $m_1(a_1, b_1) <_{C'} m_2(a_2, b_2)$, if either $m_1 <_{\mathcal{M}} m_2$, or $m_1 = m_2 \wedge a_1 <_{\mathcal{D}} a_2$, or $m_1 = m_2 \wedge a_1 = a_2 \wedge b_1 <_{\mathcal{D}} b_2$, (2) for each $m \in M$ and $a, b \in D$, $0 <_{C'} 1 <_{C'} m(a, b)$.

- $\mathrm{QLTS}(\mathrm{Spec}_{\mathrm{emp}}) = (\{q_0\}, \mathrm{Op}(\mathcal{M}, \mathcal{D}), \{q_0\} \times \mathrm{Op}(\mathcal{M}, \mathcal{D}) \times \{q_0\}, q_0, C', f')$, where transition cost function $f' : \{q_0\} \times \mathrm{Op}(\mathcal{M}, \mathcal{D}) \times \{q_0\} \rightarrow C'$ maps transition $(q_0, m(a, b), q_0)$ to $m(a, b)$ for each operation $m(a, b)$.

Then, we determine costs of traces by path cost function $g' : \mathrm{qtr}(\mathrm{Spec}_{\mathrm{emp}}) \rightarrow C'$ as follows: Given a sequence $s = (o_1, o_1) \cdot \ldots \cdot (o_l, o_l)$, g' maps s to 0, if there exists a sequence $s_1 \in \mathrm{Op}(\mathcal{M}, \mathcal{D})^*$ and a sequence $s_2 \in \mathrm{Spec}$, such that $o_1 \cdot \ldots \cdot o_l$ is a prefix of s_1, and for each d in domain of Q, $\mathrm{distance}(s_1|_d, s_2|_d) \leqslant Q(d)$. Otherwise, g' maps s to 1.

It is obvious that g' maps $(o_1, o_1) \cdot \ldots \cdot (o_l, o_l)$ to 0, if and only if $o_1 \cdot \ldots \cdot o_l \in Q$-Spec(Spec). According to definition of quasi-linearizability and k-linearizability, it is easy to see that the 0-relaxed specification $\mathrm{Spec}_{\mathrm{emp}0}$ is equivalent to the the Q-quasi-sequential specification Q-Spec(Spec). Therefore, we reduce the problem of whether a concurrent data structure is quasi-linearizable with respect to regular specification into the problem of whether a concurrent data structure is k-linearizable with respect to regular specification. The undecidability of k-linearizability is then obvious from the undecidability of quasi-linearizabilility in Lemma 2.

In [11], we reduce the k-Z decision problem of a counter machine [9], a known undecidable problem, into a quasi-linearizability problem. Then, by Theorem 1, it can be seen that the k-Z decision problem of a counter machine can also be reduced into a k-linearizability problem.

5 Decidability of regular-relaxed linearizability

The undecidability of k-linearizability is due to less restriction on cost functions (or relaxation strategies). In contrast, relaxation strategies of regular-relaxed linearizability are simple and regular. For example, a specification automaton can only relax $\mathrm{spec}_{\mathrm{emp}}$ into a regular relaxed specification. Hence, there is no specification automaton that can act as path cost function g' in the proof of Theorem 1.

Let $|S|$ be the number of elements of a set S. The following lemma shows that if the set of traces of $\mathrm{QLTS}_r(\mathrm{Spec})$ is regular, then the regular-relaxed specification $\mathrm{Spec}_{\mathcal{A}}$ is regular, and the number of its states is exponential to the number of states of $\mathcal{A}$ and polynomial to that of $\mathrm{QLTS}_r(\mathrm{Spec})$ and the number of operations.

Lemma 3. If the set of traces of $\mathrm{QLTS}_r(\mathrm{Spec})$ is regular, then the regular-relaxed specification $\mathrm{Spec}_{\mathcal{A}}$ is regular, and can be represented by a deterministic finite state automaton which has at most $g \times 2^h$ states and $g^2 \times 2^{2h} \times |\mathrm{Op}(\mathcal{M}, \mathcal{D})|$ transitions, where g and h are the number of states of $\mathrm{QLTS}_r(\mathrm{Spec})$ and $\mathcal{A}$, respectively.

Proof. Assume that $\mathrm{QLTS}_r(\mathrm{Spec}) = (Q_S, \Sigma, Q_S \times \Sigma \times Q_S, q_{s0}, \mathbb{N} \cup \{\infty\}, f)$ and $\mathcal{A} = (Q_{\mathcal{A}}, F_{\mathcal{A}}, \Sigma_{\mathcal{A}}, \rightarrow_{\mathcal{A}}, q_{\mathcal{A}0})$. Since set of traces of $\mathrm{QLTS}_r(\mathrm{Spec})$ is regular, the domain of cost is finite and let it be C_{fin}. We prove this lemma by constructing a deterministic finite automaton Aut that accepts $\mathrm{Spec}_{\mathcal{A}}$. Aut $= (Q, F, q_{\mathrm{in}}, \Sigma, \rightarrow)$ and is constructed by subsect construction as follows:

- Q is the finite set of states. Each state $q \in Q$ is a tuple of a subset of $Q_{\mathcal{A}}$ and a state in Q_S. $F = Q$ is the set of accepting states. $q_{in} = (\{q_{\mathcal{A}0}\}, q_{s0})$ is the initial state.

- $\Sigma = \mathrm{Op}(\mathcal{M}, \mathcal{D})$ is the finite set of transition labels of Aut.

- $\rightarrow \subset Q \times \Sigma \times Q$ is the transition relation. For state $(\{q_1^{\mathcal{A}}, \ldots, q_u^{\mathcal{A}}\}, q^s) \in Q$ and transition label $m(a,b) \in \mathrm{Op}(\mathcal{M}, \mathcal{D})$, $(\{q_1'^{\mathcal{A}}, \ldots, q_v'^{\mathcal{A}}\}, q'^s)$ is a successor from $(\{q_1^{\mathcal{A}}, \ldots, q_u^{\mathcal{A}}\}, q^s)$ with an $m(a,b)$ transition, if the following three conditions hold.

(1) There exists a cost $r \in C_{\mathrm{fin}}$, such that $q^s \xrightarrow{(m(a,b),r)} q'^s$ be a transition of $\mathrm{QLTS}_r(\mathrm{Spec})$.

(2) For each $1 \leqslant i \leqslant u$, if there exists state q', predicates po and pc, such that $q_i^{\mathcal{A}} \xrightarrow{(\mathrm{po},\mathrm{pc})}_{\mathcal{A}} q'$ is a transition in $\mathcal{A}$, $m(a,b)$ satisfies po and r satisfies pc, then $q' = q_j'^{\mathcal{A}}$ for some $1 \leqslant j \leqslant v$.

(3) For each $1 \leqslant j \leqslant v$, if there exists state q, predicates po and pc, such that $q \xrightarrow{(\mathrm{po},\mathrm{pc})}_{\mathcal{A}} q_j'^{\mathcal{A}}$ is a transition in $\mathcal{A}$, $m(a,b)$ satisfies po and r satisfies pc, then $q = q_i^{\mathcal{A}}$ for some $1 \leqslant i \leqslant u$.

We now prove that the language of Aut is a subset of $\mathrm{Spec}_{\mathcal{A}}$. For a sequence $s = o_1 \cdot \ldots \cdot o_l$ in language of Aut, there exist consecutive transitions $p_1 \xrightarrow{o_1}_{\mathcal{A}} p_2, \ldots, p_l \xrightarrow{o_l}_{\mathcal{A}} p_{l+1}$ of Aut with $p_1 = (\{q_{\mathcal{A}0}\}, q_{s0})$. Assume that for each i, $p_i = ((q_1^i, \ldots, q_{u_i}^i), q_s^i)$. According to the construction of Aut, there exist costs $r_1, \ldots, r_l \in C_{\mathrm{fin}}$ and states $q_{\mathrm{ind}_1}'^1, \ldots, q_{\mathrm{ind}_{l+1}}'^{l+1}$ of $\mathcal{A}$, such that for each i, (1) $q_{\mathrm{ind}_i}'^i \in \{q_1^i, \ldots, q_{u_i}^i\}$, (2) $q_s^i \xrightarrow{(o_i,r_i)} q_s^{i+1}$ is a transition of $\mathrm{QLTS}_r(\mathrm{Spec})$, (3) there exist predicates po_i and pc_i, such that $q_{\mathrm{ind}_i}'^i \xrightarrow{(\mathrm{po}_i,\mathrm{pc}_i)}_{\mathcal{A}} q_{\mathrm{ind}_{i+1}}'^{i+1}$ is a transition in $\mathcal{A}$, o_i satisfies po_i and r_i satisfies pc_i. Note that $q_{\mathrm{ind}_1}'^1 = q_{\mathcal{A}0}$ and $q_s^1 = q_{s0}$. Therefore, $s \in \mathrm{Spec}_{\mathcal{A}}$.

We then prove that $\mathrm{Spec}_{\mathcal{A}}$ is a subset of the language of Aut. For a sequence $s = o_1 \cdot \ldots \cdot o_l \in \mathrm{Spec}_{\mathcal{A}}$, since $s \in \mathrm{Spec}_{\mathcal{A}}$, there exist costs $r_1, \ldots, r_l \in C_{\mathrm{fin}}$, states $q_s^1, \ldots, q_s^{l+1}$ of $\mathrm{QLTS}_r(\mathrm{Spec})$ and states $q_{\mathcal{A}}^1, \ldots, q_{\mathcal{A}}^{l+1}$ of $\mathcal{A}$ such that $q_s^1 \xrightarrow{(o_1,r_1)}_{\mathcal{A}} q_s^2 \xrightarrow{(o_2,r_2)}_{\mathcal{A}} \cdots q_s^l \xrightarrow{(o_l,o_l)}_{\mathcal{A}} q_s^{l+1}$ is a path of $\mathrm{QLTS}_r(\mathrm{Spec})$, and there exist consecutive transitions $q_{\mathcal{A}}^1 \xrightarrow{(\mathrm{po}_1,\mathrm{pc}_1)}_{\mathcal{A}} q_{\mathcal{A}}^2, \ldots, q_l \xrightarrow{(\mathrm{po}_l,\mathrm{pc}_l)}_{\mathcal{A}} q_{\mathcal{A}}^{l+1}$ of $\mathcal{A}$ from the initial state $q_{\mathcal{A}0} \equiv q_{\mathcal{A}}^1$, such that for each $1 \leqslant i \leqslant l$, o_i satisfies po_i, r_i satisfies pc_i. Then, it is easy to see that there exists transitions $(S_1, q_s^1) \xrightarrow{o_1} (S_2, q_s^2), \ldots, (S_l, q_s^l) \xrightarrow{o_l} (S_{l+1}, q_s^{l+1})$ of Aut, and for each $1 \leqslant i \leqslant l+1$, $q_{\mathcal{A}}^i \in S_i$. Therefore, s is in the language of Aut.

We have proved that Aut accepts $\mathrm{Spec}_{\mathcal{A}}$, and Aut contains at most $g \times 2^h$ states and $g^2 \times 2^{2h} \times |\mathrm{Op}(\mathcal{M}, \mathcal{D})|$ transitions. This completes the proof of this lemma.

Based on Lemmas 1 and 3, it is obvious that the problem of regular-relaxed linearizability is decidable to concurrent data structure, when the set of traces of $\mathrm{QLTS}_r(\mathrm{Spec})$ is regular and the number of processes is bounded, as shown by the following theorem.

Theorem 2 (Decidability of regular-relaxed linearizability)**.** Given a concurrent data structure $\mathcal{L}$, a specification automaton $\mathcal{A}$ and $\mathrm{QLTS}_r(\mathrm{Spec})$, whose set of traces is regular, the problem of whether $\mathcal{L}$ is regular-relaxed linearizable with respect to Spec for n processes is decidable.

In [8], we construct specification automata for relaxation strategies of typical relaxed queues and stacks. These specification automata has the following two features:

- The transition is "deterministic". That is, for each state q of specification automaton, each operation $m(a, b)$ and each cost r, there is at most one transitions $q \xrightarrow{(\mathrm{po},\mathrm{pc})}_{\mathcal{A}} q'$ in specification automaton, such that $m(a, b)$ satisfies po and r satisfies pc.
- For each transition label $(\mathrm{po}, \mathrm{pc})$ of specification automaton, the predicates pc and pc can be decided in constant $O(1)$ time.

From the proof of Lemma 3, it is easy to see that such specification automaton $\mathrm{Spec}_{\mathcal{A}}$ can be represented by a deterministic finite state automaton Aut, and the number of states of $Q_{\mathcal{A}}$ in each state of Aut is always 1. Therefore, such Aut contains less states and transitions, and can be generated in bounded time. That is, Aut has at most $g \times h$ states and $g^2 \times h^2 \times |\mathrm{Op}(\mathcal{M}, \mathcal{D})|$ transitions, and can be constructed in $O(g^2 \times h^2 \times |\mathrm{Op}(\mathcal{M}, \mathcal{D})|)$ time. Here g and h are the number of states of $\mathrm{QLTS}_r(\mathrm{Spec})$ and $\mathcal{A}$, respectively. In this way, we reduce a regular-relaxed linearizability problem into a linearizability problem, while checking linearizability with respect to a regular sequential specification has been proved to be EXPSPACE-complete [17].

6 Conclusion and future work

Many relaxed data structures have been developed for the sake of performance and scalability. Quasi-linearizability, k-linearizability and regular-relaxed linearizability are quantitative relaxation variants of linearizability, which relax sequential specifications in a quantitative way. Relaxations by these variants can cover typical relaxed data structures, but inevitably raise their complexity for verification.

We have proven that both quasi-linearizability and k-linearizability are undecidable for a bounded number of processes. This reveals that these two variants of linearizability permit relaxations that are too complex to be verified. In contrast, we have proven that regular-relaxed linearizability is decidable for a bounded number of process. Hence, it can indeed balance the expressiveness of a relaxation with the complexity in verifying the correctness of a relaxed data structure. Therefore, regular-relaxed linearizability is more suitable for verification. Moreover, we have shown that the number of the states of a relaxed specification is exponential to the number of the states of the underlying specification automaton, but polynomial to the number of the states of the underlying quantitative sequential specification and the number of operations. However, for typical relaxed queues and stacks, the number of the states of a relaxed specification is polynomial to the number of the states of the underlying quantitative sequential specification and specification automaton and to the number of operations. Therefore, the verification of typical relaxed data structures can be effectively handled in practice.

It is interesting, as future work, to develop effective tools for verifying regular-relaxed linearizability, as in [18, 19]. We would also like to study how to use static analysis techniques, such as [20–22], to verify regular-relaxed linearizability with a unbounded number of memory locations and within a unbounded data domain.

Acknowledgements This work was partially supported by National Natural Science Foundation of China (Grants Nos. 61672504, 60721061, 60833001, 61572478, 61672503, 61100069, 61161130530) and National Basic Research Program of China (973 Program) (Grant No. 2014CB340700).

Conflict of interest The authors declare that they have no conflict of interest.

References

1 Haas A, Lippautz M, Henzinger T A, et al. Distributed queues in shared memory: multicore performance and scalability through quantitative relaxation. In: Proceedings of Computing Frontiers Conference, Ischia, 2013. 1–9

2 Kirsch C M, Lippautz M, Payer H. Fast and scalable, lock-free k-fifo queues. In: Proceedings of the 12th International Conference on Parallel Computing Technologies, St. Petersburg, 2013. 208–223

3 Kirsch C M, Payer H, Röck H, et al. Performance, scalability, and semantics of concurrent FIFO queues. In: Proceedings of the 12th International Conference on Algorithms and Architectures for Parallel Processing, Fukuoka, 2012. 273–287

4 Wimmer M, Versaci F, Träff J L, et al. Data structures for task-based priority scheduling. In: Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Orlando, 2014. 379–380

5 Herlihy M P, Wing J M. Linearizability: a correctness condition for concurrent objects. ACM Trans Program Lang Syst, 1990, 12: 463–492

6 Afek Y, Korland G, Yanovsky E. Quasi-linearizability: relaxed consistency for improved concurrency. In: Proceedings of the 14th International Conference on Principles of Distributed Systems, Tozeur, 2010. 395–410

7 Henzinger T A, Kirsch C M, Payer H, et al. Quantitative relaxation of concurrent data structures. In: Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Rome, 2013. 317–328

8 Wang C, Lv Y, Wu P. Decomposable relaxation for concurrent data structures. In: Proceedings of the 43th International Conference on Current Trends in Theory and Practice of Computer Science. Berlin: Springer-Verlag, 2017. 188–202

9 Alur R, McMillan K, Peled D. Model-checking of correctness conditions for concurrent objects. Inf Comput, 2000, 160: 167–188

10 Bouajjani A, Emmi M, Enea C, et al. Verifying concurrent programs against sequential specifications. In: Proceedings of the 22nd European Conference on Programming Languages and Systems. Berlin: Springer, 2013. 290–309

11 Wang C, Lv Y, Liu G, et al. Quasi-linearizability is undecidable. In: Proceedings of the 13th Asian Symposium on Programming Languages and Systems. Berlin: Springer, 2015. 369–386

12 Adhikari K, Street J, Wang C, et al. Verifying a quantitative relaxation of linearizability via refinement. In: Proceedings of the 20th International Symposium on Model Checking Software, New York, 2013. 24–42

13 Zhang L, Chattopadhyay A, Wang C. Round-up: runtime checking quasi linearizability of concurrent data structures. In: Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering, Silicon Valley, 2013. 4–14

14 Bouajjani A, Emmi M, Enea C, et al. Tractable refinement checking for concurrent objects. In: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Mumbai, 2015. 651–662

15 Cerný P, Radhakrishna A, Zufferey D, et al. Model checking of linearizability of concurrent list implementations. In: Proceedings of the 22nd International Conference on Computer Aided Verification. Berlin: Springer, 2010. 465–479

16 Wang C, Lv Y, Wu P. Bounded TSO-to-SC linearizability is decidable. In: Proceedings of the 42th International Conference on Current Trends in Theory and Practice of Computer Science. Berlin: Springer-Verlag, 2016. 404–417

17 Hamza J. On the complexity of linearizability. In: Proceedings of the International Conference on NETworkedked sYStems, Agadir, 2015. 308–321

18 Burckhardt S, Dern C, Musuvathi M, et al. Line-up: a complete and automatic linearizability checker. In: Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation, Toronto, 2010. 330–340

19 Liu Y, Chen W, Liu Y A, et al. Verifying linearizability via optimized refinement checking. IEEE Trans Softw Eng, 2013, 39: 1018–1039

20 Amit D, Rinetzky N, Reps T, et al. Comparison under abstraction for verifying linearizability. In: Proceedings of the 19th International Conference on Computer Aided Verification, Berlin, 2007. 477–490

21 Berdine J, Lev-Ami T, Manevich R, et al. Thread quantification for concurrent shape analysis. In: Proceedings of the 20th International Conference on Computer Aided Verification, Princeton, 2008. 399–413

22 Vafeiadis V. Automatically proving linearizability. In: Proceedings of the 22nd International Conference on Computer Aided Verification. Berlin: Springer, 2010. 450–464