Refinement Checking Parameterised Quorum Systems*

Antti Siirtola
M3S research unit, Faculty of Information Technology and Electrical Engineering, University of Oulu
Email: firstname.lastname@oulu.fi

***Abstract*—Many fault-tolerant algorithms are based on decisions made by a quorum of nodes. Since the algorithms are utilised in safety critical applications such as distributed databases, it is necessary to make sure that they operate reliably under every possible scenario. We introduce a generic compositional formalism, based on parameterised labelled transition systems, which allows us to express safety properties of parameterised quorum systems. We prove that any parameterised verification task expressible in the formalism collapses into finitely many finite state refinement checking problems. The technique is implemented in a tool, which performs the verification completely automatically. As an example, we prove the leader election phase of the Raft consensus algorithm correct for an arbitrary number of terms and for a cluster of any size.**

1. Introduction

Many fault-tolerant distributed algorithms are based on decisions made by a quorum of nodes. Consensus and atomic broadcast protocols, Paxos [1], Zab [2], and Raft [3], are examples of such algorithms. They are used in distributed databases and key-value stores, which are from time to time under a heavy load. In these circumstances, seemingly rare combinations of events become frequent [4], which implies that it is necessary to make sure that the algorithms operate reliably under every possible scenario. Consequently, the verification of fault-tolerant distributed systems has gained a lot of attention recently [5], [6], [7], [8], [9], [10], [11].

We introduce a formal theory for modelling and analysing *quorum systems*. The key characteristics of such systems is that their topology evolves over time and connectivity is defined by *quorum sets*, subsets which cover more than a half of the base set, which can be, e.g., the set of all servers. As a running example, we consider the leader election phase of the Raft consensus algorithm [3]. In Raft, time is divided into *terms* of arbitrary length, numbered with consecutive integers, and a server can crash at any moment. When a server is running, it is in one of three states, a follower, candidate, or leader. A server always (re)starts as a follower. A follower can vote for at most one server in a term. If a follower does not regularly receive messages from the leader, it increases its term and promotes itself to a candidate. A candidate sends vote requests to the other servers and if it receives a quorum of votes, it becomes a leader. However, if a server learns that another server has a higher term, it updates its term and reverts back to the follower. Our goal is to formally prove that in each term, there is at most one leader independent of the number of terms and the size of the cluster.

Technically, our approach is based on parameterised labelled transition systems (PLTSs) [12], where a system implementation and specification are basically expressed as the parallel composition of labelled transition systems (LTSs). On the implementation side, one can use hiding, too, and correctness is understood as trace refinement, which allows for the analysis of safety properties. The current theory allows for compositional reasoning and lends support to multiple parameters controlling the number of replicated LTSs. It also allows for the specification of the system topology by using the universal fragment of first order logic. This includes systems with, e.g., a star, bipartite, and totally (un)connected topology. However, systems with a quorum topology are not supported since in order to specify a quorum set, we should be able to say that for each element not in the set, there is a unique element in the set. This requires the use of existential quantification, which makes the verification problem undecidable [12]. Another possibility to model quorum sets is to use a counter over replicated components, but also this quickly leads to undecidability.

In this paper, we equip the PLTS formalism with parameters of a new kind, *quorum function variables (QFVs)*, while maintaining compositionality. QFVs can be thougth as functions from tuples of the ids of replicated components to quorum sets of the ids of replicated components. By using a QFV which assigns a quorum set for each server in each term, we can model, e.g., the leader election phase of Raft where the connectivity of the servers may change from term to term. After that, we prove that for each verification task expressible in our formalism, there are upper bounds, *cut-*

∗. A. Siirtola: Refinement Checking Parameterised Quorum Systems. In A. Legay, K. Schneider, eds.: 17th International Conference on Application of Concurrency to System Design (ACSD), pp. 39–48, Zaragoza, Spain, June 28–30, 2017. IEEE, 2017. Available: https://doi.org/10.1109/ACSD.2017.15

offs, to the parameters such that if the system meets the specification for all parameter values up to the cut-offs, then the system is correct with respect to the specification for all parameter values. The cut-offs are computable and often very tight, because they are not shared by all verification tasks expressible in the formalism as in [13], but determined for each implementation and specification process separately. After obtaining the cut-offs, the verification can be completed by using efficient, existing tools for finite state systems. The whole process is implemented by extending Bounds tool, earlier introduced in [14]. In order to handle the QFVs, the results of [12] are not only adapted to take QFVs into account but several completely new results (Proposition 21, Lemma 23, Proposition 24, and Proposition 26) on the cut-offs of quorum sets are introduced.

The unique features of our formalism are its ability to express parameterised quorum systems, the support for compositional reasoning, and decidability. We are not aware of other formalisms with all these features.

In terms of compositional parameterised verification, the closest related works are the data independence (DI) results [15], the behavioural fixed point (BFP) technique [16] (similar to the network invariant method [17]), and the combination of these, the data independent induction [18]. DI applies to systems with parameterised data types but a fixed number of processes and the BFP technique can handle systems with a linear topology, whereas our focus is on systems with an arbitrary number of processes with a quorum topology. Actually, both the techniques are already combined with our original verification technique without the QFVs in [19], [20], but the same extensions should work with our QFV-enabled formalism, too. That is because the BFP technique is applied before and the DI results after our approach. Our verification technique is previously adapted to modal interface automata [21], too, but neither this formalism lends support to the analysis of systems with a quorum topology.

From the viewpoint of quorum systems, the closest related work is probably the parametric interval abstraction (PIA) [8]. PIA applies to parameterised systems with threshold transitions, which can be used to model quorum systems. However, the technique does not lend support to compositional reasoning and since it is abstraction-based, termination is not guaranteed. In [9], quorum transitions are introduced in order to treat a parameterised Paxos protocol, but this approach only enables verification with respect to some parameters. Theorem proving techniques, which typically involve some degree of user intervention, for the formal verification of the Raft protocol are considered in [10], [11], and the efficient verification of the small instances of parameterised quorum systems in [5], [6], [7].

There are also several other approaches to parameterised verification based on cut-offs [13], [22], induction [16], [17], [18], abstraction [23], [24], [25], and infinite state verification algorithms [26], [27]. However, most cut-off results and infinite state verification algorithms do not lend support to quorum topologies or topologies that evolve over time because it often breaks decidability. On the other hand, induction and abstraction methods are typically incomplete and not guaranteed to terminate.

In the next section, we recall the calculus of LTSs. After that, we equip LTSs with parameters and show that the resulting parameterised formalism is compositional. In Section 4, we present our main result, the cut-off theorem for PLTSs with QFVs. The paper concludes with discussion on future work.

2. Labelled Transition Systems

In this section, we briefly recall a CSP-like LTS-based process calculus with parallel composition and hiding operators and trace refinement [28]. Basically, the only difference with the usual LTS notation is that events have an explicit data part which makes adding parameterisation convenient.

We assume that there is a countably infinite set of *events*. One of them is the *invisible* event, denoted τ, and the other events are *visible*. The visible events have an explicit channel and data part; we assume countably infinite sets $\mathbb{C}$ and $\mathbb{A}$ of respectively *channels* and *constants* and that each visible event is of the form $c(\mathbf{e})$, where c is a channel and $\mathbf{e}$ is a finite tuple of constants.

A *labelled transition system (LTS)* is a four-tuple $L := (S, E, R, \dot{s})$, where (1) S is a non-empty set of *states*, (2) E is a set of visible events, (3) $R \subseteq S \times (E \cup \{\tau\}) \times S$ is a set of *transitions*, and (4) $\dot{s}$ is the *initial* state. The second component E is called *the alphabet (of L)*.

Let L_i be an LTS $(S_i, E_i, R_i, \dot{s}_i)$ for both $i \in \{1, 2\}$. The *parallel composition (of L_1 and L_2)* is an LTS

$$L_1 \parallel L_2 := (S_1 \times S_2, E_1 \cup E_2, R_\parallel, (\dot{s}_1, \dot{s}_2)) \,,$$

where $R_\parallel$ is the set of all triples $((s_1, s_2), \alpha, (s_1', s_2'))$ such that either (1) $\alpha \neq \tau$ and $(s_i, \alpha, s_i') \in R_i$ for both $i \in \{1, 2\}$; (2) $(s_1, \alpha, s_1') \in R_1$, $\alpha \notin E_2$, $s_2 \in S_2$ and $s_2' = s_2$; or (3) $(s_2, \alpha, s_2') \in R_2$, $\alpha \notin E_1$, $s_1 \in S_1$ and $s_1' = s_1$.

Let L be an LTS $(S, E, R, \dot{s})$ and E' a set of visible events. The LTS L *after hiding* E' is an LTS $L \setminus E' := (S, E \setminus E', R_\setminus, \dot{s})$, where $R_\setminus$ is the set of (1) all triples $(s, \alpha, s') \in R$ such that $\alpha \notin E'$; and (2) all triples (s, τ, s') such that $(s, \alpha, s') \in R$ for some $\alpha \in E'$.

A finite alternating sequence $(s_0, \alpha_1, s_1, \ldots, \alpha_n, s_n)$ of states and events of L is an *execution of* L if s_0 is the initial state and (s_{i-1}, α_i, s_i) is a transition of L for every $i \in \{1, \ldots, n\}$. A finite sequence of visible events is a *trace (of L)*, if there is an execution of L such that the sequence can be obtained from the execution by erasing all the states and the invisible events. The set of all the traces of L is denoted by $\mathrm{tr}(L)$. An LTS L_1 is a *trace refinement* of an LTS L_2, denoted $L_1 \preceq_{\mathrm{tr}} L_2$, if L_1 and L_2 have the same alphabet and $\mathrm{tr}(L_1) \subseteq \mathrm{tr}(L_2)$ [29]. The LTSs L_1 and L_2 are *trace equivalent*, denoted $L_1 \equiv_{\mathrm{tr}} L_2$, if and only if $L_1 \preceq_{\mathrm{tr}} L_2$ and $L_2 \preceq_{\mathrm{tr}} L_1$. Clearly, $\preceq_{\mathrm{tr}}$ is a preorder (i.e., a reflexive and transitive relation) and $\equiv_{\mathrm{tr}}$ an equivalence relation on the set of LTSs.

The operators and the trace refinement relation have many useful properties [12], [28], [29], which are exploited

in the proofs. The parallel composition is commutative, associative, and idempotent with respect to $\equiv_{\mathrm{tr}}$ (i.e., $L \parallel L \equiv_{\mathrm{tr}} L$ for all LTSs L) and a single-state LTS $L_{id} := (\{\dot{s}\}, \emptyset, \emptyset, \dot{s})$ with the empty alphabet and no transition is the identity element of $\parallel$. This allows us to extend $\parallel$ to every finite set $I = \{i_1, \ldots, i_n\}$ and all LTSs $L_{i_1}, \ldots, L_{i_n}$ by defining

$$\parallel_{i \in I} L_i := \begin{cases} L_{i_1} \parallel (\parallel_{i \in I \setminus \{i_1\}} L_i), & \text{when } n > 0\,, \\ L_{id}, & \text{when } n = 0\,. \end{cases}$$

Moreover, distributing hiding over parallel composition results in an LTS greater in the preorder; $(L_1 \parallel L_2) \setminus E \preceq_{\mathrm{tr}} (L_1 \setminus E) \parallel (L_2 \setminus E)$ for all LTSs L_1, L_2 and every set E of visible events [12]. Finally, $\preceq_{\mathrm{tr}}$ is compositional with respect to the parallel composition and hiding operators; if $L_1 \preceq_{\mathrm{tr}} L_2$, then $L_1 \parallel L_3 \preceq_{\mathrm{tr}} L_2 \parallel L_3$ and $L_1 \setminus E \preceq_{\mathrm{tr}} L_2 \setminus E$ for all LTSs L_1, L_2, L_3 and every set E of visible events.

3. Parameterised Labelled Transition Systems

In this section, we equip LTSs with parameters while preserving compositionality. This will be done in the same fashion as in [12], but here we also introduce parameters of a new kind, quorum function variables.

Definition 1 (Quorum set)**.** Let M be a subset of B. The set M is a *quorum set (of B)* if and only if $2|M| > |B|$.

We write $\mathbb{P}_q(B)$ for the set of all the quorum sets of B, i.e., the set of all subsets of B which cover more than half of B, and $\mathbb{P}_{\mathrm{qe}}(B)$ for the set $\mathbb{P}_q(B) \cup \{\emptyset\}$, which includes the empty set, too.

In order to model Raft, we parameterise LTSs and operators with *types*, *variables*, and *quorum function variables (QFVs)*. A type represents a finite non-empty set of constants that are typically used as the identifiers of replicated components of a certain kind. A variable represents a single constant, i.e., it typically refers to an individual component of a certain type. A QFV represents a function which maps a tuple of constants to a quorum set of a certain type or to the empty set. Hence, a QFV typically maps a tuple of component ids to a quorum set of component ids or to the empty set. The sets of types, variables, and QFVs are denoted by $\mathbb{T}$, $\mathbb{X}$, and $\mathbb{F}$, respectively.

Formally, we assume that (1) for each type T there is a countably infinite set $I_T \subseteq \mathbb{A}$ denoting the elements of the type T such that I_U and I_V are disjoint for different types U and V, (2) for each variable x there is a type T_x determining the domain of x, and (3) for each QFV Π there is a type T_Π denoting the image of Π and a (possibly empty) tuple $\mathbf{T}_\Pi := (T_\Pi^1, \ldots, T_\Pi^{n_\Pi})$ of types denoting the domain of Π. The possible values of types, variables, and QFVs are determined by a valuation.

Definition 2 (Valuation)**.** A *valuation* is a function ϕ whose domain is a finite set of types, variables, and QFVs such that

1) for each type $T \in \mathrm{dom}(\phi)$, $\phi(T)$ is a finite non-empty subset of I_T,
2) for every variable $x \in \mathrm{dom}(\phi)$, $T_x \in \mathrm{dom}(\phi)$ and $\phi(x) \in \phi(T_x)$, and
3) for every QFV $\Pi \in \mathrm{dom}(\phi)$, $T_\Pi, T_\Pi^1, \ldots, T_\Pi^{n_\Pi} \in \mathrm{dom}(\phi)$ and $\phi(\Pi)$ is a function: $\phi(T_\Pi^1) \times \ldots \times \phi(T_\Pi^{n_\Pi}) \to \mathbb{P}_{\mathrm{qe}}(\phi(T_\Pi))$.

A valuation is extended to sets and tuples of types, variables, and QFVs in the usual way, by applying it to each tuple componentwise and to each set elementwise. Note that the value of a QFV may be a function that maps some of its arguments to the empty set instead of a quorum set. This is needed to carry out the proof of Proposition 18.

Example 3. For our Raft model, we pick a type T_S to represent the set of the identifiers of servers and a type T_T to represent the set of the identifiers of terms. We also use a QFV Π_S with $T_{\Pi_S} = T_S$ and $\mathbf{T}_{\Pi_S} = (T_S, T_T)$ to assign each server and each term a set of servers from which the server needs a vote in order to become a leader in the term. Variables x_0, x_1, and x_2 of the type T_S are used to refer to individual servers and a variable y of the type T_T is used to refer to a specific term.

If we want to consider an instance of Raft with three servers and a single term, we can use, for instance, a valuation ϕ such that $\phi(T_S) = \{s_1, s_2, s_3\}$, $\phi(T_T) = \{t_1\}$, $\phi(x_0) = s_1$, $\phi(x_1) = \phi(x_2) = s_2$, $\phi(y) = t_1$, and $\phi(\Pi_S) = \{((s_1, t_1), \{s_1, s_2\}), ((s_2, t_1), \{s_2, s_3\}), ((s_3, t_1), \emptyset)\}$. Note that $\phi(\Pi_S)$ maps, e.g., a tuple (s_1, t_1) to $\{s_1, s_2\}$, which is a quorum set of $\phi(T_S)$. In the context of our model, it means that in order for the server s_1 to become a leader in the term t_1, it needs a vote from itself and the server s_2. On the other hand, (s_3, t_1) is mapped to the empty set, which means that the server s_3 should not be able to become a leader in the term t_1.

We use guards to represent (i) (in)equality tests between variables and (ii) inclusion tests between a variable and a quorum set. Formally, a *guard* G is a propositional formula which is formed from atomic expressions of the form $\top$ (true), $x = y$ (equality test), $x \neq y$ (inequality test), $z \in \Pi(x_1, \ldots, x_{n_\Pi})$ (inclusion test), and binary connectives $\vee$ (or) and $\wedge$ (and). Above, x, y are variables such that $T_x = T_y$, Π is a QFV, z a variable such that $T_z = T_\Pi$, and $x_1, \ldots, x_{n_\Pi}$ are variables such that $\mathbf{T}_\Pi = (T_{x_1}, \ldots, T_{x_{n_\Pi}})$.

The *parameters* of a guard G are the variables and the QFVs occurring in G. The set of all the parameters of G is denoted by $\mathrm{par}(G)$ and

$$\overline{\mathrm{par}}(G) := \mathrm{par}(G) \cup \{T_x \mid x \in \mathrm{par}(G) \cap \mathbb{X}\} \cup \{T_\Pi, T_\Pi^1, \ldots, T_\Pi^{n_\Pi} \mid \Pi \in \mathrm{par}(G) \cap \mathbb{F}\}$$

is the set of all the parameters of G plus the types of the parameters. A valuation ϕ is said to be *compatible (with G)* if and only if $\mathrm{par}(G) \subseteq \mathrm{dom}(\phi)$.

Let G be a guard and ϕ a compatible valuation. The *ϕ-instance of G* or the *instance of G (generated by ϕ)*, denoted by $[\![G]\!]_\phi$, is the value of the proposition which is obtained from G by substituting *true* for $\top$, $\phi(x)$ for every variable x, and $\phi(\Pi)$ for every QFV Π occurring in G.

Example 4. Let us consider a guard $G := (x_0 \neq x_1) \wedge (x_1 \in \Pi_S(x_0, y))$, where Π_S is a QFV and x_0, x_1, and y are the

variables of Example 3. The guard represents a claim: "the server x_0, which is different from x_1, needs a vote from x_1 in order to become a leader in the term y." The guard has four parameters, namely the variables x_0, x_1, y and the QFV Π_S, which implies that $\overline{\mathrm{par}}(G) = \{x_0, x_1, y, \Pi_S, T_T, T_S\}$. Let ϕ be the valuation of Example 3. Obviously, ϕ is compatible with G and since $\phi(x_0) \neq \phi(x_1)$ and $\phi(x_1) \in \phi(\Pi_S)(\phi(x_0), \phi(y))$, $[\![G]\!]_\phi$ evaluates to *true*.

A structure $c(\mathbf{x})$, where c is a channel and $\mathbf{x}$ a tuple of variables, is a *parameterised visible event*. The simplest PLTSs are basically LTSs where parameterised visible events are substituted for the ordinary ones.

Definition 5 (EPLTS)**.** An *elementary parameterised LTS (EPLTS)* is a four-tuple $(S, \Sigma, \Delta, \dot{s})$, where (1) S is a finite non-empty set of *states*, (2) Σ is a finite set of parameterised visible events, (3) $\Delta \subseteq S \times (\Sigma \cup \{\tau\}) \times S$ is a finite set of *parameterised transitions*, and (4) $\dot{s}$ is the *initial* state.

Example 6. In order to model the Raft specification, we first consider it from the viewpoint of two servers, x_0 and x_1, and a term y. We use a parameterised event $leader(x_i, y)$, where $i \in \{0, 1\}$, to denote that the server x_i is chosen as a leader in the term y. Here, *leader* is a channel and x_0, x_1, and y are variables of the type T_S, T_S, and T_T, respectively.

Now, the specification from the viewpoint of two servers and a term can be formalised as an EPLTS $Spec2(x_0, x_1, y)$ in Figure 1, which formally says that no two servers can become a leader during the same term but repeating a leader announcement is fine.

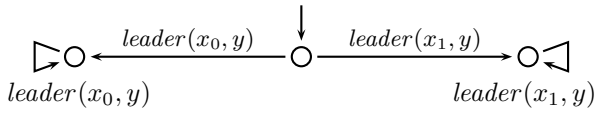

Figure 1. EPLTS $Spec2(x_0, x_1, y)$ representing the Raft specification from the viewpoint of two servers, x_0 and x_1, and a term y

More complicated PLTSs are constructed from elementary ones by using guards and (replicated) parallel composition and hiding operators.

Definition 7 (PLTS)**.** *Parameterised LTSs (PLTSs)* are generated inductively as follows:

1) An EPLTS is a PLTS.
2) If P is a PLTS and G a guard, then $([G]P)$ is a PLTS.
3) If P_1 and P_2 are PLTSs, then $(P_1 \parallel P_2)$ is a PLTS.
4) If P is a PLTS and C a finite set of channels, then $(P \setminus C)$ is a PLTS.
5) If P is a PLTS and x a variable, then $(\|_x P)$ is a PLTS.

A variable x is *bound* in a PLTS P if it occurs in P and its every occurrence is within a substructure $\|_x P'$ of P. The other variables occurring in P are *free* in P. The *parameters* of P are the free variables, the types of bound variables, and the QFVs occurring in P. Now, the sets $\mathrm{par}(P)$ and $\overline{\mathrm{par}}(P)$ and the notion of compatibility can be defined like in the case of guards: $\mathrm{par}(P)$ is the set of all parameters of P and $\overline{\mathrm{par}}(P)$ is the set of all parameters of P plus the types of the parameters. We may also write $P(\mathbf{x}, \mathbf{T}, \mathbf{\Pi})$ to emphasise that P is a PLTS the parameters of which are the variables in the tuple $\mathbf{x}$, the types in the tuple $\mathbf{T}$, and the QFVs in the tuple $\mathbf{\Pi}$. A valuation ϕ is *compatible* with P if and only if $\mathrm{par}(P) \subseteq \mathrm{dom}(\phi)$.

Each PLTS represents (infinitely) many LTSs obtained by fixing the values of parameters and evaluating the operators. Let ϕ be a valuation and x a variable such that $T_x \in \mathrm{dom}(\phi)$. We write $\mathrm{ext}(\phi, x)$ for the set of all valuations ϕ' with the domain $\mathrm{dom}(\phi) \cup \{x\}$ such that $\phi'(x) \in \phi(T_x)$ and $\phi'|_{\mathrm{dom}(\phi)\setminus\{x\}} = \phi|_{\mathrm{dom}(\phi)\setminus\{x\}}$, i.e., ϕ and ϕ' agree on the values of parameters outside x.

Definition 8 (Instance of a PLTS)**.** Let P be a PLTS and ϕ a compatible valuation. The *ϕ-instance of P* or the *instance of P (generated by ϕ)* is denoted by $[\![P]\!]_\phi$ and determined inductively as follows:

1) $[\![(S, \Sigma, \Delta, \dot{s})]\!]_\phi = (S, \{c(\phi(\mathbf{x}))|c(\mathbf{x}) \in \Sigma\}, \{(s, c(\phi(\mathbf{x})), s')|(s, c(\mathbf{x}), s') \in \Delta\} \cup \{(s, \tau, s') \in \Delta\}, \dot{s})$,
2) $[\![[G]P']\!]_\phi = \begin{cases} [\![P']\!]_\phi, & \text{if } [\![G]\!]_\phi \text{ is } \mathit{true}, \\ L_{id}, & \text{if } [\![G]\!]_\phi \text{ is } \mathit{false}, \end{cases}$
3) $[\![P_1 \parallel P_2]\!]_\phi = [\![P_1]\!]_\phi \parallel [\![P_2]\!]_\phi$,
4) $[\![P' \setminus C]\!]_\phi = [\![P']\!]_\phi \setminus \{c(a_1, \ldots, a_n) \mid c \in C, a_1, \ldots, a_n \in \mathbb{A}\}$,
5) $[\![\|_x P']\!]_\phi = \|_{\phi' \in \mathrm{ext}(\phi, x)} [\![P']\!]_{\phi'}$.

We can also write $P(\phi(\mathbf{x}), \phi(\mathbf{T}), \phi(\mathbf{\Pi}))$ for the ϕ-instance of a PLTS $P(\mathbf{x}, \mathbf{T}, \mathbf{\Pi})$.

Example 9. Recall the EPLTS *Spec2* of Figure 1 representing the partial Raft specification. As we let the variable y to range over all term identifiers and x_0 and x_1 over all servers identifiers, we obtain the model of the full specification

$$Spec := \|_{x_0} \|_{x_1} \|_{y} [x_0 \in \Pi_S(x_0, y) \wedge x_1 \in \Pi_S(x_1, y)] Spec2(x_0, x_1, y) \ ,$$

which says that for each term, there can be at most one leader. Moreover, the guard guarantees that the leader is a server that voted for itself in this term. Note that since *Spec2* has no bound variable, $\mathrm{par}(Spec2) = \{x_0, x_1, y\}$ but $\overline{\mathrm{par}}(Spec2) = \{x_0, x_1, y, T_S, T_T\}$. On the other hand, *Spec* has no free variable. That is why $\mathrm{par}(Spec) = \overline{\mathrm{par}}(Spec) = \{\Pi_S, T_S, T_T\}$.

In order to visualize *Spec*, let us consider a valuation ϕ such that $\phi(T_T) = \{t_1\}$, $\phi(T_S) = \{s_1, \ldots, s_n\}$ and $s_i \in \phi(\Pi_S)(s_i, t_1)$ for all $i \in \{1, \ldots, n\}$. Obviously, the valuation is compatible with *Spec* and the ϕ-instance of *Spec* is a star-shaped LTS in Figure 2, which indeed says that there can be at most one leader for the term t_1.

We complete our parameterised formalism by extending the trace refinement relation to the set of PLTSs while preserving compositionality.

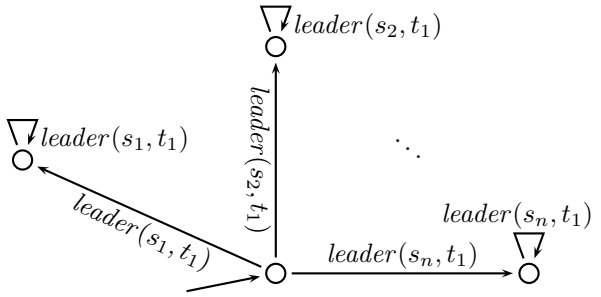

Figure 2. The instance of *Spec* representing the Raft specification from the viewpoint of n servers $s_1, \ldots, s_n$ and a term t_1

Definition 10 (Trace refinement on PLTSs)**.** A PLTS P_1 is a *trace refinement* of a PLTS P_2, denoted $P_1 \preceq_{\mathrm{tr}} P_2$, if and only if $[\![P_1]\!]_\phi \preceq_{\mathrm{tr}} [\![P_2]\!]_\phi$ for all compatible valuations ϕ.

Proposition 11. *The trace refinement relation $\preceq_{\mathrm{tr}}$ is a compositional preorder on the set of PLTSs.*

Parameterised verification tasks can be now expressed with the aid of PLTSs and the relation above. Given a system implementation PLTS P and a system specification PLTS Q, we consider P to be *correct (with respect to Q)* if and only if $P \preceq_{\mathrm{tr}} Q$.

Example 12. In order to model the Raft implementation, we use a parameterised event $vote(x_0, y, x_1)$ to denote that the server x_0 votes for the server x_1 in the term y and a parameterised event $candidate(x_0, y)$ to denote that the server x_0 promotes itself to a candidate in the term y.

The behaviour of the implementation of Raft is modelled in the same fashion as the specification. First, we capture it in the follower/candidate mode from the viewpoint of servers x_0, x_1, x_2 and a term y in an EPLTS *Flw3* in Figure 3. The EPLTS says that in the term y, the server x_0 can vote for either x_1 or x_2, but not both, or become a candidate. When we let the variables x_1, x_2, and y to range over all values in their domain (with the restriction that the values of x_1 and x_2 are different), we obtain the model of a single server x_0 running in the follower mode as the PLTS

$$Flw := \mathop{\|}_{x_1}\mathop{\|}_{x_2} [x_1 \neq x_2] \mathop{\|}_{y} Flw3(x_0, x_1, x_2, y) \ ,$$

which formally states that a server can vote for at most one server in the term or become a candidate.

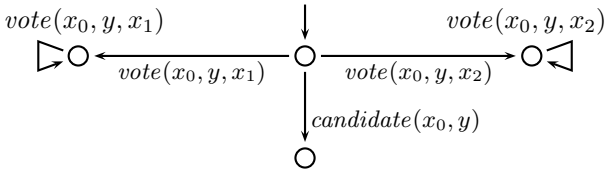

Figure 3. EPLTS $Flw3(x_0, x_1, x_2, y)$ representing the Raft implementation in the follower/candidate mode from the viewpoint of servers x_0, x_1, x_2 and a term y

Second, we model the Raft implementation in the candidate/leader mode from the viewpoint of servers x_0, x_1 and a term y as an EPLTS *Ldr2* in Figure 4. This model says that once the server x_0 becomes a candidate and receives a vote from the server x_1, it can promote itself to a leader in the term y. As we let y to range over all term ids and x_1 to range over all values in a quorum set of the server x_0 for the term y, the model of a single server x_0 running in the leader mode is obtained as the PLTS

$$Ldr := \mathop{\|}_{y}\mathop{\|}_{x_1} [x_0 \in \Pi_S(x_0, y) \wedge x_1 \in \Pi_S(x_0, y)] Ldr2 \ ,$$

which says that in order for a server to become a leader, it needs to become a candidate and then receive a vote from a quorum of servers, including itself.

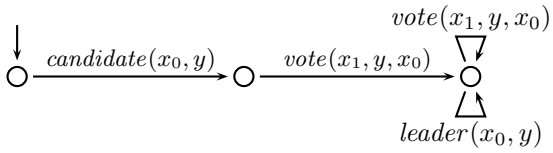

Figure 4. EPLTS $Ldr2(x_0, x_1, y)$ representing the Raft implementation in the candidate/leader mode from the viewpoint of servers x_0, x_1 and a term y

When we compose the partial models in parallel and let x_0 to range over all server ids, we obtain the model of the Raft implementation with an arbitrary many servers and terms as the PLTS

$$Raft := \mathop{\|}_{x_0} (Ldr \parallel Flw) \ .$$

In order to analyse the correctness of Raft, we hide the events which are irrelevant to the specification. We write LE for the set $\{vote, candidate\}$ and $Raft'$ for the PLTS $Raft \setminus LE$. Now, the problem whether the protocol operates correctly can be formalised as the question whether $Raft' \preceq_{\mathrm{tr}} Spec$ holds.

Note that in our model, a quorum set needed by a server to become a leader can change from term to term. Hence, the formalism enables the specification of parameterised systems where the topology evolves over time but for each instance, the changes in the topology are fixed a priori. This is not a restriction though since every possible combination of topology changes is covered by some valuation. Hence, every trace of the protocol is contained in some instance of the model, which is sufficient for the analysis of safety properties.

4. Refinement Checking PLTSs

In this section, we show how to compute structural cut-offs for types such that to prove a parameterised system correct for all the parameter values it is sufficient to check only finitely many instances up to the cut-offs, provided the specification does not involve hiding. The requirement on not using hiding in specifications is necessary for decidability [12]. However, the restriction is not severe since hiding is typically only applied on the implementation side. Hence, from now on, *an implementation PLTS* refers to any PLTS,

whereas *a specification PLTS* means a PLTS which does not involve hiding.

Intuitively, the main steps of our reduction technique are the following:

1) First, we show that if a big instance of the implementation PLTS P (resp., a specification PLTS Q) is composed of the same components as a set of small instances and each small instance of P is a trace refinement of the corresponding instance of Q, then the big instance of P is a trace refinement of the big instance of Q, too (Proposition 18).
2) Second, we prove that there is an upper limit for the size of small instances which is obtained by a structural analysis of the system implementation and specification (Propositions 21 and 26).
3) By combining the above results, we see that a trace refinement checking task on PLTSs reduces to finitely many refinement checking tasks on finite state LTSs (Theorem 28).

Next, we present the technique formally.

Definition 13 (Set of the processes of an instance)**.** Let R be a PLTS and ϕ a compatible valuation. *The set of the processes (of the ϕ-instance of R)*, denoted by $\mathrm{prc}(R,\phi)$, is defined inductively as follows:

1) $\mathrm{prc}(R,\phi) = \{[\![R]\!]_\phi\}$, when R is an EPLTS,
2) $\mathrm{prc}(R_1 \parallel R_2,\phi) = \bigcup_{i\in\{1,2\}} \big(\{i\} \times \mathrm{prc}(R_i,\phi)\big)$,
3) $\mathrm{prc}(R \setminus C,\phi) = \mathrm{prc}(R,\phi)$,
4) $\mathrm{prc}([G]R',\phi) = \begin{cases} \mathrm{prc}(R',\phi), & \text{if } [\![G]\!]_\phi \text{ is } \textit{true},\\ \emptyset, & \text{otherwise,}\end{cases}$
5) $\mathrm{prc}(\|_x R',\phi) = \bigcup_{\phi'\in\mathrm{ext}(\phi,x)} \{\phi'(x)\} \times \mathrm{prc}(R',\phi')$.

Example 14. Let θ be a valuation with the domain $\{T_S, T_T, \Pi_S\}$ such that $\theta(T_S) = \{s_1,\dots,s_n\}$, $\theta(T_T) = \{t_1,\dots,t_m\}$, and for all $i \in \{1,\dots,n\}$ and all $k \in \{1,\dots,m\}$, $\theta(\Pi_S)(s_i,t_k)$ is a quorum set of $\theta(T_S)$ such that $s_i \in \theta(\Pi_S)(s_i,t_k)$. Then $\mathrm{prc}(\mathit{Spec},\theta)$ is

$$\bigcup_{i=1}^{n}\bigcup_{j=1}^{n}\bigcup_{k=1}^{m} \{(s_i,(s_j,(t_k,\mathit{Spec2}(s_i,s_j,t_k))))\}$$

and $\mathrm{prc}(\mathit{Raft}',\theta)$ is

$$\bigcup_{i=1}^{n}\bigcup_{k=1}^{m}\bigcup_{\substack{j=1\\ s_j\in\phi(\Pi_S)(s_i,t_k)}}^{n} \{(s_i,(1,(t_k,(s_j,\mathit{Ldr2}(s_i,s_j,t_k)))))\} \cup$$

$$\bigcup_{i=1}^{n}\bigcup_{\substack{j=1\\ j\neq k}}^{n}\bigcup_{k=1}^{n}\bigcup_{l=1}^{m} \{(s_i,(2,(s_j,(s_k,(t_l,\mathit{Flw3}(s_i,s_j,s_k,t_l))))))\}\ .$$

Definition 15 (Subvaluation)**.** Let Γ be a set of types. A valuation ϕ_1 is a Γ-*subvaluation* of a valuation ϕ_2 if and only if the valuations have the same domain,

1) $\phi_1(T) \subseteq \phi_2(T)$ for all types $T \in \Gamma$,
2) $\phi_1|_{\mathbb{X}} = \phi_2|_{\mathbb{X}}$ and $\phi_1|_{\mathbb{T}\setminus\Gamma} = \phi_2|_{\mathbb{T}\setminus\Gamma}$, that is, ϕ_1 and ϕ_2 agree on the values of the variables and the other types in the domain, and
3) for all QFVs $\Pi \in \mathrm{dom}(\phi_1)$ and for all $(c_1,\dots,c_n) \in \mathrm{dom}(\phi_1(\Pi))$, $\phi_1(\Pi)(c_1,\dots,c_n) = \phi_2(\Pi)(c_1,\dots,c_n) \cap \phi_1(T_\Pi)$, provided it is a quorum set of $\phi_1(T_\Pi)$, otherwise $\phi_1(\Pi)(c_1,\dots,c_n) = \emptyset$.

We say that the ϕ_1-instance of a PLTS R is *smaller than (or equal to)* the ϕ_2-instance of R if and only if ϕ_1 is a $\mathbb{T}$-subvaluation of ϕ_2.

Example 16. Let θ be as in Example 14 and Θ the set of all valuations θ' with the domain $\{T_S, T_T, \Pi_S\}$ such that $\theta'(T_T) = \{t_l\}$ and $\theta'(T_S) = \{s_i, s_j, s_k\}$ for some $l \in \{1,\dots,m\}$ and $i,j,k \in \{1,\dots,n\}$ and for all $s \in \theta'(T_S)$, $\theta'(\Pi_S)(s,t_l) = \theta(\Pi_S)(s,t_l) \cap \{s_i,s_j,s_k\}$, provided it is a quorum set of $\{s_i,s_j,s_k\}$, otherwise $\theta'(\Pi_S)(s,t_l) = \emptyset$. Then Θ is a finite set of $\mathbb{T}$-subvaluations of θ and $[\![\mathit{Spec}]\!]_{\theta'}$ is smaller than $[\![\mathit{Spec}]\!]_\theta$ for all $\theta' \in \Theta$.

Since a valuation and its subvaluations map the variables in the same way, it is obvious that an instance of an EPLTS equals the instance of the EPLTS generated by a smaller valuation. A similar result holds for guards as well.

Lemma 17. *Let R be an EPLTS, G a guard, T a type, and ψ,ϕ valuations compatible with R and G such that ϕ is a $\{T\}$-subvaluation of ψ. Then the following holds.*

1) $[\![R]\!]_\psi = [\![R]\!]_\phi$.
2) *If $[\![G]\!]_\psi$ is false, then $[\![G]\!]_\phi$ is false.*
3) *If $[\![G]\!]_\psi$ is true and $\phi(\Pi)(\phi(\mathbf{x})) = \emptyset$ if and only if $\psi(\Pi)(\psi(\mathbf{x})) = \emptyset$ whenever $\Pi(\mathbf{x})$ with $T_\Pi = T$ occurs in G, then $[\![G]\!]_\phi$ is true.*

Proof. 1. Since $\phi|_{\mathbb{X}} = \psi|_{\mathbb{X}}$ and an instance of R is completely defined by the values of free variables, the claim is evident. 2. & 3. By induction on the structure of G by using the claim as the induction hypothesis. □

Recall that specification PLTSs Q do not involve hiding. It implies that each instance of Q is just the parallel composition of the LTSs, which are the instances of the EPLTSs occurring in Q. Hence, if the set of the processes of a (big) instance $[\![Q]\!]_\psi$ equals the set of the processes of (smaller) instances in $\{[\![Q]\!]_\phi \mid \phi \in \Phi\}$, then by the commutativity, associativity, and idempotence of the parallel composition, the big instance is trace equivalent to the parallel composition of the small instances, i.e., $\|_{\phi\in\Phi}[\![Q]\!]_\phi \equiv_{\mathrm{tr}} [\![Q]\!]_\psi$. For an implementation PLTS P, which may involve hiding, we can show a weaker result: if the set of the processes of a (big) instance $[\![P]\!]_\psi$ equals the set of the processes of the (smaller) instances in $\{[\![P]\!]_\phi \mid \phi \in \Phi\}$, the big instance is a trace refinement of the parallel composition of the small instances, i.e., $[\![P]\!]_\psi \preceq_{\mathrm{tr}} \|_{\phi\in\Phi}[\![P]\!]_\phi$. By the compositionality of the trace refinement, this implies that we can derive the correctness of a big system instance from the correctness of small instances.

Proposition 18. *Let P be an implementation PLTS and Q a specification PLTS, ψ a valuation compatible with both the PLTSs, and Φ a finite set of $\mathbb{T}$-subvaluations of ψ such that $\mathrm{prc}(P,\psi) = \bigcup_{\phi\in\Phi}\mathrm{prc}(P,\phi)$ and $\mathrm{prc}(Q,\psi) =$*

$\bigcup_{\phi\in\Phi}\mathrm{prc}(Q,\phi)$. *If* $[\![P]\!]_\phi \preceq_{\mathrm{tr}} [\![Q]\!]_\phi$ *for all valuations* $\phi\in\Phi$, *then* $[\![P]\!]_\psi \preceq_{\mathrm{tr}} [\![Q]\!]_\psi$.

Proof. Similar to the proof of Proposition 27 in [12]. □

Example 19. Let θ and Θ be as in Examples 14 and 16. Since every element of $\mathrm{prc}(\mathit{Spec},\theta)$ depends on the identifiers of two servers and one term, it is easy to see that $\mathrm{prc}(\mathit{Spec},\theta)$ equals $\bigcup_{\theta'\in\Theta}\mathrm{prc}(\mathit{Spec},\theta')$, i.e., the θ-instance of *Spec* is composed of the same components as the set of θ'-instances, where $\theta'\in\Theta$. Similarly, we can see that every element of $\mathrm{prc}(\mathit{Raft}',\theta)$ depends on the identifiers of at most three servers and one term, which implies that $\mathrm{prc}(\mathit{Raft}',\theta)$ equals $\bigcup_{\theta'\in\Theta}\mathrm{prc}(\mathit{Raft}',\theta')$. Since Θ is finite, by Proposition 18, it means that if $[\![\mathit{Raft}']\!]_{\theta'} \preceq_{\mathrm{tr}} [\![\mathit{Spec}]\!]_{\theta'}$ for all $\theta'\in\Theta$, then $[\![\mathit{Raft}']\!]_\theta \preceq_{\mathrm{tr}} [\![\mathit{Spec}]\!]_\theta$, too.

The proposition allows us to discard (big) instances but it does not clearly say which instances we should keep. This piece of information is hidden in the condition which requires the set of the processes of the big instance to be the same as the set of the processes of the small ones. Since each element in $\mathrm{prc}(R,\psi)$ depends on the values of finitely many variables as well as the values of finitely many QFVs, the size of the valuations we should keep is determined by the number of free and bound variables and the number of QFVs in R.

To put it more formally, we write $\mathrm{free}_T(R)$ for the set $\{x\in\mathbb{X}\cap\mathrm{par}(R)\mid T_x=T\}$ of the free variables of the type T occurring in R, $\mathrm{bnd}_T(R)$ for the maximum number of the nested bound variables of the type T in R, formally defined as

1) $\mathrm{bnd}_T(R)=0$ for an EPLTS R,
2) $\mathrm{bnd}_T([G]R')=\mathrm{bnd}_T(R'\setminus C)=\mathrm{bnd}_T(R')$,
3) $\mathrm{bnd}_T(R_1\parallel R_2)=\max(\mathrm{bnd}_T(R_1),\mathrm{bnd}_T(R_2))$,
4) $\mathrm{bnd}_T(\|_x R')=\mathrm{bnd}_T(R')+\begin{cases}1, & \text{if } T_x=T,\\ 0, & \text{if } T_x\neq T,\end{cases}$

and $\deg_T(R)$ for the maximum number of the structures of the form $\Pi(\mathbf{x})$ in a branch of the syntax tree of R such that $T_\Pi=T$, formally defined as

1) $\deg_T(R)=0$ for an EPLTS R,
2) $\deg_T([G]R')=\deg_T(G)+\deg_T(R')$, where
   - $\deg_T(G)$ is the number of structures $\Pi(\mathbf{x})$ in G such that $T_\Pi=T$ (multiple occurrences of the same structure $\Pi(\mathbf{x})$ are not counted),
3) $\deg_T(R_1\parallel R_2)=\max(\deg_T(R_1),\deg_T(R_2))$,
4) $\deg_T(R'\setminus C)=\deg_T(\|_x R')=\deg_T(R')$.

Example 20. Recall the Raft specification *Spec*. It involves three substructures of the form $\|_x R'$, one with $T_x=T_T$ and two with $T_x=T_S$, which means that $\mathrm{bnd}_{T_T}(\mathit{Spec})=1$ and $\mathrm{bnd}_{T_S}(\mathit{Spec})=2$. Similarly, there are two structures of the form $\Pi(\mathbf{x})$, both with $T_\Pi=T_S$, which gives $\deg_{T_T}(\mathit{Spec})=0$ and $\deg_{T_S}(\mathit{Spec})=2$. Since *Spec* has no free variable, $\mathrm{free}_{T_T}(\mathit{Spec})=\mathrm{free}_{T_S}(\mathit{Spec})=\emptyset$.

As regards the Raft implementation *Raft′*, it has six substructures of the form $\|_x R'$. For $T_x=T_S$, three of them are nested and for $T_x=T_T$, only one of them is nested, which gives $\mathrm{bnd}_{T_T}(\mathit{Raft}')=1$ and $\mathrm{bnd}_{T_S}(\mathit{Raft}')=3$. Moreover, as only one structure of the form $\Pi(\mathbf{x})$ occurs in *Raft′* and in this case $T_\Pi=T_S$, we get $\deg_{T_T}(\mathit{Raft}')=0$ and $\deg_{T_S}(\mathit{Raft}')=1$. Since *Raft′* has no free variable, $\mathrm{free}_{T_T}(\mathit{Raft}')=\mathrm{free}_{T_S}(\mathit{Raft}')=\emptyset$.

Proposition 21. *Let* R *be a PLTS,* ψ *a valuation compatible with* R, T *a type in* $\mathrm{dom}(\psi)$, *and* Φ *a set of* $\{T\}$*-subvaluations of* ψ. *Moreover, let us assume that for every* $F\subseteq\psi(T)$ *such that* $\psi(\mathrm{free}_T(R))\subseteq F$ *and* $|F|\leq|\mathrm{free}_T(R)|+\mathrm{bnd}_T(R)$ *and for all quorum sets* $M_1,\ldots,M_k$ *of* $\psi(T)$ *such that* $k\leq\deg_T(R)$, *there is* $\phi\in\Phi$ *such that* $F\subseteq\phi(T)$ *and* $M_1\cap\phi(T),\ldots,M_k\cap\phi(T)$ *are quorum sets of* $\phi(T)$. *Then* $\mathrm{prc}(R,\psi)=\bigcup_{\phi\in\Phi}\mathrm{prc}(R,\phi)$.

Proof. We argue by induction on the structure of R by using the lemma as an induction hypothesis.

In the base step, R is an EPLTS. First, note that Φ is non-empty. Secondly, since each $\phi\in\Phi$ is a subvaluation of ψ, by Lemma 17, it means that $[\![R]\!]_\psi=[\![R]\!]_\phi$ for every $\phi\in\Phi$. Then, by definition, it is evident that $\mathrm{prc}(R,\psi)=\bigcup_{\phi\in\Phi}\mathrm{prc}(R,\phi)$.

In the induction step, there are four cases to consider.

1. First, let us assume that R is $[G]R'$. **1.1** If $[\![G]\!]_\psi$ is *false*, then by Lemma 17, $[\![G]\!]_\phi$ is *false* for all $\phi\in\Phi$. It implies that then $\mathrm{prc}(R,\psi)=\emptyset=\bigcup_{\phi\in\Phi}\mathrm{prc}(R,\phi)$. **1.2** Let us then assume that $[\![G]\!]_\psi$ is *true* and let Φ_t be the set of valuations $\phi\in\Phi$ such that $[\![G]\!]_\phi$ is *true*. Let $F'\subseteq\psi(T)$ such that $\psi(\mathrm{free}_T(R'))\subseteq F'$ and $|F'|\leq|\mathrm{free}_T(R')|+\mathrm{bnd}_T(R')$ and let $M'_1,\ldots,M'_{k'}$ be quorum sets of $\psi(T)$ such that $k'\leq\deg_T(R')$. Moreover, let $M_1,\ldots,M_l$ be the non-empty sets $\psi(\Pi)(\psi(\mathbf{x}))$ such that $\Pi(\mathbf{x})$ occurs in G and $T_\Pi=T$. Now, $M_1,\ldots,M_l$ are quorum sets of $\psi(T)$ and $k'+l\leq\deg_T(R')+\deg_T(G)\leq\deg_T(R)$. Since $\mathrm{free}_T(R')\subseteq\mathrm{free}_T(R)$ and $\mathrm{bnd}_T(R)=\mathrm{bnd}_T(R')$, we also know that $|F'\cup\psi(\mathrm{free}_T(R))|\leq|\mathrm{free}_T(R)|+\mathrm{bnd}_T(R)$. Hence, by assumption, there is $\phi\in\Phi$ such that $F'\cup\psi(\mathrm{free}_T(R))\subseteq\phi(T)$ and $M'_1\cap\phi(T),\ldots,M'_{k'}\cap\phi(T),M_1\cap\phi(T),\ldots,M_l\cap\phi(T)$ are quorum sets of $\phi(T)$. This means that $\phi(\Pi)(\phi(\mathbf{x}))=\emptyset$ if and only if $\psi(\Pi)(\psi(\mathbf{x}))=\emptyset$ whenever $\Pi(\mathbf{x})$ with $T_\Pi=T$ occurs in G. Since $[\![G]\!]_\psi$ is *true*, by Lemma 17, it implies that $[\![G]\!]_\phi$ is *true* as well. Hence, $\phi\in\Phi_t$ which, in turn, means that the induction hypothesis is applicable to R', ψ, and Φ_t. Therefore, $\mathrm{prc}(R',\psi)=\bigcup_{\phi\in\Phi_t}\mathrm{prc}(R',\phi)$, which implies that

$$\mathrm{prc}(R,\psi)\overset{\text{def.}}{=}\mathrm{prc}(R',\psi)\overset{\text{i.h.}}{=}\bigcup_{\phi\in\Phi_t}\mathrm{prc}(R',\phi)\overset{\text{def.}}{=}\bigcup_{\phi\in\Phi_t}\mathrm{prc}(R,\phi)=\bigcup_{\phi\in\Phi}\mathrm{prc}(R,\phi)\,.$$

2. Next, we consider the case when R is $R_1\parallel R_2$. Let $i\in\{1,2\}$ and $F_i\subseteq\psi(T)$ such that $\psi(\mathrm{free}_T(R_i))\subseteq F_i$ and $|F_i|\leq|\mathrm{free}_T(R_i)|+\mathrm{bnd}_T(R_i)$, and let $M_1,\ldots,M_k$ be quorum sets of $\psi(T)$ such that $k\leq\deg_T(R_i)$. Since $\mathrm{free}_T(R_i)\subseteq\mathrm{free}_T(R)$, $\mathrm{bnd}_T(R_i)\leq\mathrm{bnd}_T(R)$, and $\deg_T(R_i)\leq\deg_T(R)$, we know that $|F_i\cup\psi(\mathrm{free}_T(R))|\leq$

$|\mathrm{free}_T(R)|+\mathrm{bnd}_T(R)$ and $k \leq \deg_T(R)$. By assumption, it means that there is $\phi \in \Phi$ such that $F \cup \psi(\mathrm{free}_T(R)) \subseteq \phi(T)$ and $M_1 \cap \phi(T), \ldots, M_k \cap \phi(T)$ are quorum sets of $\phi(T)$. Therefore, the induction hypothesis is applicable to R_1 and R_2. Hence, $\mathrm{prc}(R_i, \psi) = \bigcup_{\phi \in \Phi} \mathrm{prc}(R_i, \phi)$ for both $i \in \{1, 2\}$, which implies that

$$\begin{aligned}\mathrm{prc}(R, \psi) &\overset{\text{def.}}{=} \bigcup_{i \in \{1,2\}} \big(\{i\} \times \mathrm{prc}(R_i, \psi)\big) \\ &\overset{\text{i.h.}}{=} \bigcup_{i \in \{1,2\}} \big(\{i\} \times \bigcup_{\phi \in \Phi} \mathrm{prc}(R_i, \phi)\big) \\ &= \bigcup_{\phi \in \Phi} \bigcup_{i \in \{1,2\}} \big(\{i\} \times \mathrm{prc}(R_i, \phi)\big) \overset{\text{def.}}{=} \bigcup_{\phi \in \Phi} \mathrm{prc}(R, \phi)\ .\end{aligned}$$

3. Let us then assume that R is $\|_x R'$. For every $\psi' \in \mathrm{ext}(\psi, x)$, let $\Phi_{\psi'}$ be the set of all valuations $\phi' \in \bigcup_{\phi \in \Phi} \mathrm{ext}(\phi, x)$ such that $\phi'(x) = \psi'(x)$. Obviously, $\Phi_{\psi'}$ is a set of subvaluations of ψ'. Let $F \subseteq \psi'(T)$ such that $\psi(\mathrm{free}_T(R')) \subseteq F$ and $|F| \leq |\mathrm{free}_T(R')| + \mathrm{bnd}_T(R')$ and let $M_1, \ldots, M_k$ be quorum sets of $\psi'(T)$ such that $k \leq \deg_T(R')$. Since $\psi(T) = \psi'(T)$ and $\mathrm{free}_T(R) \subseteq \mathrm{free}_T(R')$, then $\psi(\mathrm{free}_T(R)) \subseteq F \subseteq \psi(T)$, $M_1, \ldots, M_k$ are quorum sets of $\psi(T)$, and $k \leq \deg_T(R)$. Moreover, if $T_x \neq T$, then it is obvious that $|F| \leq |\mathrm{free}_T(R')| + \mathrm{bnd}_T(R') = |\mathrm{free}_T(R)| + \mathrm{bnd}_T(R)$. Otherwise, if $T_x = T$, then $\mathrm{bnd}_T(R) = \mathrm{bnd}_T(R') + 1$ and $\mathrm{free}_T(R') \subseteq \mathrm{free}_T(R) \cup \{x\}$, which implies that also in this case $|F| \leq |\mathrm{free}_T(R')| + \mathrm{bnd}_T(R') \leq |\mathrm{free}_T(R)| + \mathrm{bnd}_T(R)$. By assumption, it means that there is $\phi \in \Phi$ such that $F \subseteq \phi(T)$ and $M_1 \cap \phi(T), \ldots, M_k \cap \phi(T)$ are quorum sets of $\phi(T)$. Since $\psi'(\mathrm{free}_T(R')) \subseteq F \subseteq \phi(T)$, we know that $\psi'(x) \in \phi(T)$ and therefore there is $\phi' \in \mathrm{ext}(\phi, x)$ such that $\phi' \in \Phi_{\psi'}$. Since $\phi'(T) = \phi(T)$, we see that $F \subseteq \phi'(T)$ and $M_1 \cap \phi'(T), \ldots, M_k \cap \phi'(T)$ are quorum sets of $\phi'(T)$. Hence, the induction hypothesis is applicable to R', ψ', and $\Phi_{\psi'}$, which means that $\mathrm{prc}(R', \psi') = \bigcup_{\phi' \in \Phi_{\psi'}} \mathrm{prc}(R', \phi')$ for all $\psi' \in \mathrm{ext}(\psi, x)$. Therefore,

$$\begin{aligned}\mathrm{prc}(R, \psi) &\overset{\text{def.}}{=} \bigcup_{\psi' \in \mathrm{ext}(\psi, x)} \big(\{\psi'(x)\} \times \mathrm{prc}(R', \psi')\big) \\ &\overset{\text{i.h.}}{=} \bigcup_{\psi' \in \mathrm{ext}(\psi, x)} \big(\{\psi'(x)\} \times \bigcup_{\phi' \in \Phi_{\psi'}} \mathrm{prc}(R', \phi')\big) \\ &= \bigcup_{\psi' \in \mathrm{ext}(\psi, x)} \bigcup_{\phi' \in \Phi_{\psi'}} \big(\{\phi'(x)\} \times \mathrm{prc}(R', \phi')\big) \\ &= \bigcup_{\phi \in \Phi} \bigcup_{\phi' \in \mathrm{ext}(\phi, x)} \big(\{\phi'(x)\} \times \mathrm{prc}(R', \phi')\big) \\ &\overset{\text{def.}}{=} \bigcup_{\phi \in \Phi} \mathrm{prc}(R, \phi).\end{aligned}$$

4. Finally, the case when R is $R' \setminus C$ is trivial, because $\mathrm{free}_T(R) = \mathrm{free}_T(R')$, $\mathrm{bnd}_T(R) = \mathrm{bnd}_T(R')$, and $\deg_T(R) = \deg_T(R')$.

Hence, by the induction principle, the lemma is correct. □

In order to put the assumption of Proposition 21 into a more explicit form, we need a function $q : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, which can be described as follows. Let us suppose that we have n quorum sets of B, a subset F of B, and we want to reduce the size of B by removing from $B \setminus F$ as many elements as possible such that the n sets remain quorum sets. Then $q(n, |F|)$ gives an upper bound for the size of the reduced B. Consequently, q is called a quorum bound function.

Definition 22 (Quorum bound function)**.** A function $q : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is a *quorum bound function* if for all $(n, m) \in \mathbb{N} \times \mathbb{N}$ the following holds: Given a set B, quorum sets $M_1, \ldots, M_n$ of B, and a subset F of B with $|F| = m$, there is a subset B' of B such that $|B'| \leq q(n, m)$, $F \subseteq B'$, and $M_1 \cap B', \ldots, M_n \cap B'$ are quorum sets of B'.

The existence of a quorum bound function is a combinatorial problem, which resembles Ramsey theoretic questions [30] and the Zarankiewicz problem, an unsolved problem in extremal graph theory [31]. Fortunately, we can restrict our attention to minimal quorum sets and formulate the problem in terms of linear algebra. After that, a quorum bound function can be defined with the aid of the maximum of the component-wise sum of the minimal elements of a semi-linear set.

Lemma 23. *Let B be a set, $M_1, \ldots, M_n$ and $M'_1, \ldots, M'_n$ quorum sets of B such that $M'_i \subseteq M_i$ for all $i \in \{1, \ldots, n\}$, and B' a subset of B. If $M'_1 \cap B', \ldots, M'_n \cap B'$ are quorum sets of B', then $M_1 \cap B', \ldots, M_n \cap B'$ are quorum sets of B', too.*

Proof. The claim is obvious because $M'_i \cap B' \subseteq M_i \cap B'$ and $M'_i \cap B'$ is a quorum set of B' for all $i \in \{1, \ldots, n\}$. □

Proposition 24. *Let n be a non-negative integer and $\mathbf{v}_1, \ldots, \mathbf{v}_k$ the vectors in $\{-1, 1\}^n$, i.e., $k = 2^n$. Let A be the set of all vectors $(a_1, \ldots, a_k) \in \mathbb{N}^k$ such that $a_1\mathbf{v}_1 + \ldots + a_k\mathbf{v}_k = \mathbf{0}$ (n-dimensional zero vector), $A_{\min}$ the set of all the minimal elements in A, and*

$$q_{\mathrm{n}} := \max\{a_1 + \ldots + a_k \mid (a_1, \ldots, a_k) \in A_{\min}\}\ .$$

Then $q(n, m) := (m + 1) \cdot q_n - 1$ for all $m \in \mathbb{N}$.

Proof. Let B be a set, $M_1, \ldots, M_n$ quorum sets of B, and F a subset of B with $|F| = m$. By Lemma 23, we may assume that the quorum sets $M_1, \ldots, M_n$ are minimal, i.e., $2|M_i| = |B| + 1$ for all $i \in \{1, \ldots, n\}$. Let b_0 be an element not in B. Each element $b \in B \cup \{b_0\}$ can be considered as a vector $\mathbf{v}_b := (v_1, \ldots, v_n) \in \{-1, 1\}^n$, where for all $i \in \{1, \ldots, n\}$, $v_i = 1$ if and only if $b \in M_i$. Then $\sum_{b \in B \cup \{b_0\}} \mathbf{v}_b = \mathbf{0}$. Now, $B \cup \{b_0\}$ can be partitioned into sets $B_1, \ldots, B_m$ such that for all $i \in \{1, \ldots, n\}$, $\sum_{b \in B_i} \mathbf{v}_b = \mathbf{0}$ and $|B_i| \leq q_n$. Note that by König's lemma, $A_{\min}$ is finite, which implies that q_n exists. Let B' be the union of all B_i, where $i \in \{1, \ldots, n\}$, such that $B_i \cap (F \cup \{b_0\}) \neq \emptyset$. Then $F \subseteq B' \setminus \{b_0\}$ and $\sum_{b \in B'} \mathbf{v}_b = \mathbf{0}$. This implies that $M_1 \cap (B' \setminus \{b_0\}), \ldots, M_n \cap (B' \setminus \{b_0\})$ are

quorum sets of $B' \setminus \{b_0\}$ and $|B' \setminus \{b_0\}| \leq (|F|+1) \cdot q_n - 1$. Hence, $q(n, m)$ can be defined as $(m+1) \cdot q_n - 1$. □

For each n, the value of q_n can be determined by using an SMT solver for linear arithmetic. By using the SMT solver Z3 [32], we have learned that $q_0 = 0, q_1 = q_2 = 2$, $q_3 = 4, q_4 = 6$, and $q_5 = 10$. Computing $q_0, \ldots, q_4$ took less than a second whereas the computation of q_5 took an hour on Intel i7-4770 with 8GB of memory. For greater values of n, Z3 run out of memory. Fortunately, these values are sufficient for many practical verification tasks, including our Raft example, where for each type T, the sum of $\deg_T(G)$ over nested guards G is small. Nevertheless, in future, we hope to be able to prove an explicit formula for q_n.

With the aid of a quorum bound function, we can define explicit structural cut-offs for each verification task expressible in the formalism.

Definition 25 (Cut-off)**.** The *cut-off (size)* for a type T, an implementation PLTS P, and a specification PLTS Q is

$$\begin{aligned}\mathrm{co}_T(P, Q) := \max\big(1,\\ q(\deg_T(P), \mathrm{bnd}_T(P) + |\mathrm{free}_T(P \parallel Q)|),\\ q(\deg_T(Q), \mathrm{bnd}_T(Q) + |\mathrm{free}_T(P \parallel Q)|)\big)\ .\end{aligned}$$

Proposition 26. *Let P be an implementation PLTS, Q a specification PLTS, ψ a valuation with the domain $\overline{\mathrm{par}}(P \parallel Q)$, T a type in $\mathrm{dom}(\psi)$, and Φ the set of all $\{T\}$-subvaluations ϕ of ψ such that $|\phi(T)| \leq \mathrm{co}_T(P, Q)$. Then for both $R \in \{P, Q\}$ and every $F \subseteq \psi(T)$ such that $\psi(\mathrm{free}_T(R)) \subseteq F$ and $|F| \leq |\mathrm{free}_T(R)| + \mathrm{bnd}_T(R)$ and for all quorum sets $M_1, \ldots, M_k$ of $\psi(T)$ such that $k \leq \deg_T(R)$, there is $\phi \in \Phi$ such that $F \subseteq \phi(T)$ and $M_1 \cap \phi(T), \ldots, M_k \cap \phi(T)$ are quorum sets of $\phi(T)$.*

Proof. The claim follows relatively straightforwardly from the definition of a cut-off and a quorum bound function. □

Without affecting the result of verification, it is sufficient to consider valuations which are *non-isomorphic*, i.e., cannot be obtained from each other by bijective renaming of constants. We write $\phi_1 \not\simeq \phi_2$ to denote that the valuations ϕ_1 and ϕ_2 are non-isomorphic.

Definition 27 (Cut-off set)**.** Let P be an implementation PLTS and Q a specification PLTS. A set Φ of valuations is a *cut-off set (for P and Q)* if Φ is a maximal set of valuations ϕ with the domain $\overline{\mathrm{par}}(P \parallel Q)$ such that $|\phi(T)| \leq \mathrm{co}_T(P, Q)$ for every type $T \in \mathrm{dom}(\phi)$ and $\phi_1 \not\simeq \phi_2$ whenever ϕ_1 and ϕ_2 are different elements of Φ.

By combining the results above, we get the main result of the paper, which allows for reducing a trace refinement task among PLTSs to finitely many refinement checks between finite state LTSs.

Theorem 28 (Cut-off theorem)**.** *Let P be an implementation PLTS, Q a specification PLTS, and Φ a cut-off set for P and Q. Then (1) the set Φ is finite and (2) $P \preceq_{\mathrm{tr}} Q$ if and only if $[\![P]\!]_\phi \preceq_{\mathrm{tr}} [\![Q]\!]_\phi$ for all $\phi \in \Phi$.*

Proof. Obviously, the cut-off set Φ is finite. It is also evident that if $P \preceq_{\mathrm{tr}} Q$ then $[\![P]\!]_\phi \preceq_{\mathrm{tr}} [\![Q]\!]_\phi$ for all $\phi \in \Phi$.

Let us then assume that $[\![P]\!]_\phi \preceq_{\mathrm{tr}} [\![Q]\!]_\phi$ for all $\phi \in \Phi$, and let ψ be any valuation compatible with P and Q. Obviously, we may assume that $\mathrm{dom}(\psi) = \overline{\mathrm{par}}(P \parallel Q)$. By isomorphism, we know that then $[\![P]\!]_\phi \preceq_{\mathrm{tr}} [\![Q]\!]_\phi$ for all valuations ϕ such that $\mathrm{dom}(\phi) = \overline{\mathrm{par}}(P \parallel Q)$ and $|\phi(T)| \leq \mathrm{co}_T(P, Q)$ for every type $T \in \mathrm{dom}(\phi)$. By Proposition 26, we can proceed like in the proof of Theorem 4.11 in [21] and inductively apply Proposition 21 to all types $T \in \mathrm{dom}(\psi)$. Therefore, we get that $\mathrm{prc}(P, \psi) = \bigcup_{\phi \in \Phi} \mathrm{prc}(P, \phi)$ and $\mathrm{prc}(Q, \psi) = \bigcup_{\phi \in \Phi} \mathrm{prc}(Q, \phi)$, which by Proposition 18 means that $[\![P]\!]_\psi \preceq_{\mathrm{tr}} [\![Q]\!]_\psi$. Since the choice of ψ was arbitrary, this implies that $P \preceq_{\mathrm{tr}} Q$, i.e., also the opposite claim holds. □

Example 29. By Ex. 20, we can see that $\mathrm{co}_{T_S}(\mathit{Raft}', \mathit{Spec}) = 7$ and $\mathrm{co}_{T_T}(\mathit{Raft}', \mathit{Spec}) = 1$. By Theorem 28, this implies that in order to prove Raft correct for any number of servers and terms, it is sufficient to show that all its instances up to seven servers and one term behave correctly.

Theorem 28 gives rise to a completely automatic parameterised verification approach which we have implemented by extending our Bounds tool [14]. The tool inputs an implementation and specification PLTS and computes cut-offs for the types by using Theorem 28. Since the cut-offs provided by the theorem are only rough structural ones, the tool tries to improve them further by checking the assumptions of Proposition 18 up to the rough bounds and by discarding the big instances which satisfy the assumptions of the proposition. After that, Bounds produces the trace refinement checking tasks up to the improved cut-offs. Finally, the outputted finite state verification questions are solved by using the refinement checker FDR3 [33], which gives the answer to the parameterised verification task. Bounds is publicly available at [34].

Example 30. When applied to our example, Bounds first generates all non-isomorphic valuations up to the cut-offs of seven servers and one term. There are altogether 395,790 of them, since the value of the QFV Π_S can be chosen in numerous ways. Enumerating all valuations takes 47 seconds. After that, the tool applies Proposition 18, which reveals that only 18 instances have to be actually verified. These are the instances up to three servers, and this phase takes 262 seconds. Finally, by using FDR3, all remaining instances are found to be correct within two seconds. This implies that for an arbitrary number of terms and a cluster of any size, Raft operates correctly in the sense that there is at most one leader in each term. The experiment was done on an 8-thread Intel i7-4770 with 8GB of memory and details are found in the appendices.

Once we have proved that a system implementation refines its specification, we can use the specification, which is usually much smaller, in place of the system implementation in further verification efforts. This is possible since our QFV-enabled PLTS formalism is compositional.

5. Conclusions and Future Work

We have presented a formalism for expressing parameterised quorum systems and their specifications. The formalism is compositional and it is supported by a fully automated technique that reduces a parameterised verification task into finitely many finite state verification problems. The reduction is sound and complete and determined by the structure of parameterised processes. The technique is implemented in a tool and applied to prove the correctness of the leader election part of Raft.

An obvious topic for future research is discovering an explicit formula for a quorum bound function. We also aim to consider byzantine errors as well as to verify the log replication phase of Raft, i.e., to analyse the correctness of the full Raft protocol.

Acknowledgements

The author would like to thankfully acknowledge funding from the ITEA 3 APPSTACLE research program funded by Tekes (Finnish Funding Agency for Technology and Innovation). The author would also like to thank Keijo Heljanko for discussions on the Raft protocol and Juha Kortelainen, Tuukka Salmi, and Ari Vesanen for discussions on quorum sets, which led to the discovery of Proposition 24.

References

[1] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, 1998.

[2] F. P. Junqueira, B. C. Reed, and M. Serafini, "Zab: High-performance broadcast for primary-backup systems," in *DSN '11*, J. K. Muppala, Ed. IEEE, 2011, pp. 245–256.

[3] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *USENIX ATC '14*, G. Gibson and N. Zeldovich, Eds. USENIX Association, 2014, pp. 305–320.

[4] C. Newcombe, T. Rath, F. Zhang, B. Munteanu, M. Brooker, and M. Deardeuff, "How amazon web services uses formal methods," *Commun. ACM*, vol. 58, no. 4, pp. 66–73, 2015.

[5] T. Tsuchiya and A. Schiper, "Verification of consensus algorithms using satisfiability solving," *Distrib. Comput.*, vol. 23, no. 5, pp. 341–358, 2011.

[6] P. Bokor, J. Kinder, M. Serafini, and N. Suri, "Efficient model checking of fault-tolerant distributed protocols," in *DSN '11*, J. K. Muppala, Ed. IEEE, 2011, pp. 73–84.

[7] A. John, I. Konnov, U. Schmid, H. Veith, and J. Widder, "Towards modeling and model checking fault-tolerant distributed algorithms," in *SPIN '13*, ser. LNCS, E. Bartocci and C. R. Ramakrishnan, Eds. Springer, 2013, vol. 7976, pp. 209–226.

[8] ——, "Parameterized model checking of fault-tolerant distributed algorithms by abstraction." in *FMCAD '13*, B. Jobstman and S. Ray, Eds. IEEE, 2013, pp. 201–209.

[9] G. Delzanno, M. Tatarek, and R. Traverso, "Model checking Paxos in Spin," in *GandALF '14*, ser. EPTCS, A. Peron and C. Piazza, Eds., 2014, vol. 161, pp. 131–146.

[10] J. R. Wilcox, D. Woos, P. Panchekha, Z. Tatlock, X. Wang, M. D. Ernst, and T. Anderson, "Verdi: A framework for implementing and formally verifying distributed systems," in *PLDI '15*, D. Grove and S. Blackburn, Eds. ACM, 2015, pp. 357–368.

[11] C. Hawblitzel, J. Howell, M. Kapritsos, J. R. Lorch, B. Parno, M. L. Roberts, S. Setty, and B. Zill, "Ironfleet: Proving practical distributed systems correct," in *SOSP '15*, E. L. Miller and S. Hand, Eds. ACM, 2015, pp. 1–17.

[12] A. Siirtola and J. Kortelainen, "Multi-parameterised compositional verification of safety properties," *Inform. Comput.*, vol. 244, pp. 23–48, 2015.

[13] B. Aminof, T. Kotek, S. Rubin, F. Spegni, and H. Veith, "Parameterized model checking of rendezvous systems," in *CONCUR '14*, ser. LNCS, P. Baldan and D. Gorla, Eds. Springer, 2014, vol. 8704, pp. 109–124.

[14] A. Siirtola, "Bounds2: A tool for compositional multi-parametrised verification," in *TACAS '14*, ser. LNCS, E. Ábrahám and K. Havelund, Eds. Springer, 2014, vol. 8413, pp. 599–604.

[15] R. S. Lazić, "A semantic study of data independence with applications to model checking," Ph.D. dissertation, Oxford University, 1999.

[16] A. Valmari and M. Tienari, "An improved failures equivalence for finite-state systems with a reduction algorithm," in *PSTV '91*, B. Jonsson, J. Parrow, and B. Pehrson, Eds. North-Holland, 1991, pp. 3–18.

[17] R. P. Kurshan and K. L. McMillan, "A structural induction theorem for processes," *Inform. Comput.*, vol. 117, no. 1, pp. 1–11, 1995.

[18] S. J. Creese, "Data independent induction: CSP model checking of arbitrary sized networks," Ph.D. dissertation, Oxford University, 2001.

[19] A. Siirtola, "Cut-offs with network invariants," in *ACSD '10*, L. Gomez, V. Khomenko, and J. Fernandes, Eds. IEEE, 2010, pp. 105–114.

[20] A. Siirtola and K. Heljanko, "Parametrised compositional verification with multiple process and data types," in *ACSD '13*, J. Carmona, M. T. Lazarescu, and M. Pietkiewicz-Koutny, Eds. IEEE, 2012, pp. 67–76.

[21] ——, "Parametrised modal interface automata," *ACM Trans. Embed. Comput. Syst.*, vol. 14, no. 4, pp. 65:1–65:25, 2015.

[22] Q. Yang and M. Li, "A cut-off approach for bounded verification of parameterized systems," in *ICSE '10*, J. Kramer, J. Bishop, P. T. Devanbu, and S. Uchitel, Eds. ACM, 2010, pp. 345–354.

[23] S. K. Lahiri, R. E. Bryant, and B. Cook, "A symbolic approach to predicate abstraction," in *CAV '03*, ser. LNCS, W. A. Hunt Jr. and F. Somenzi, Eds. Springer, 2003, vol. 2725, pp. 141–153.

[24] B. Wachter and B. Westphal, "The spotlight principle," in *VMCAI '07*, ser. LNCS, B. Cook and A. Podelski, Eds. Springer, 2007, vol. 4349, pp. 182–198.

[25] E. M. Clarke, M. Talupur, and H. Veith, "Proving ptolemy right: The environment abstraction framework for model checking concurrent systems," in *TACAS '08*, ser. LNCS, C. R. Ramakrishnan and J. Rehof, Eds. Springer, 2008, vol. 4963, pp. 33–47.

[26] J. D. Bingham and A. J. Hu, "Empirically efficient verification for a class of infinite-state systems," in *TACAS '05*, ser. LNCS, N. Halbwachs and L. D. Zuck, Eds. Springer, 2005, vol. 3440, pp. 77–92.

[27] S. Ghilardi and S. Ranise, "Backward reachability of array-based systems by SMT solving: termination and invariant synthesis," *Log. Meth. Comput. Sci.*, vol. 6, no. 4, pp. 1–48, 2010.

[28] A. W. Roscoe, *Understanding Concurrent Systems*. Springer, 2010.

[29] C. A. R. Hoare, *Communicating sequential processes*. Prentice-Hall, 1985.

[30] S. Fujita, C. Magnant, and K. Ozeki, "Rainbow generalizations of Ramsey theory: A survey," *Graph. Combinator.*, vol. 26, no. 1, pp. 1–30, 2010.

[31] Z. Füredi and M. Simonovits, "The history of degenerate (bipartite) extremal graph problems," in *Erdős Centennial*, ser. Bolyai Society Mathematical Studies, L. Lovász, I. Z. Ruzsa, and V. T. Sós, Eds. Springer, 2013, pp. 169–264.

[32] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *TACAS '08*, ser. LNCS, C. R. Ramakrishnan and J. Rehof, Eds., vol. 4963. Springer, 2008, pp. 337–340.

[33] T. Gibson-Robinson, P. Armstrong, A. Boulgakov, and A. W. Roscoe, "FDR3: A parallel refinement checker for CSP," *STTT*, vol. 18, no. 2, pp. 149–167, 2016.

[34] A. Siirtola, "Bounds website," 2016, http://cc.oulu.fi/~asiirtol/bounds.

Appendix A.
Bounds Code for Raft Leader Election

```
type S
type T

chan vote : S, T, S
chan candidate : S, T
chan leader : S, T

qfv Maj : S <- S,T

var s0 : S
var s1 : S
var s2 : S
var t : T

plts Spec2 =
    lts
        I   =       leader(s0,t) -> S0
                []  leader(s1,t) -> S1
        S0  =       leader(s0,t) -> S0
        S1  =       leader(s1,t) -> S1
    from I

plts Spec = || s0,s1,t : [s0 in Maj(s0,t) & s1 in Maj(s1,t)] Spec2

plts Ldr2 =
    lts
        C   =       candidate(s0,t) -> C1
        C1  =       vote(s1,t,s0) -> L
        L   =       leader(s0,t) -> L
                []  vote(s1,t,s0) -> L
    from C

plts Flw3 =
    lts
        F   =       candidate(s0,t) -> STOP
                []  vote(s0,t,s1) -> F1
                []  vote(s0,t,s2) -> F2
        F1  =       vote(s0,t,s1) -> F1
        F2  =       vote(s0,t,s2) -> F2
    from F

plts Raft = || s0 : ((|| t,s1 : [s0 in Maj(s0,t) & s1 in Maj(s0,t)] Ldr2 )
                 || (|| s1, s2 : [!s1=s2] || t: Flw3))

pset LE = (_) s0,s1,t: {candidate(s0,t), vote(s0,t,s1)}

trace refinement: verify Raft \ LE against Spec
```

Appendix B.
Run of Bounds on Raft Leader Election

```
This is Bounds 2.4!
Created by Antti Siirtola 2010-2016 (contact: antti.siirtola@oulu.fi)

Reducing a parameterised trace refinement task to a finitary one.
(implementation: (Raft\LE), specification: Spec)

Determining a cut-off (size) for the type S...
...Succeeded. The cut-off size is 7.
Press enter to accept it or provide another value to change it
```

```
Determining a cut-off (size) for the type T...
...Succeeded. The cut-off size is 1.
Press enter to accept it or provide another value to change it

Computing the values of parameters...
    96%   #val:562211       #can:562098       #sto:55487

.
.
.

...Done!

Found 395790 non-isomorphic valuations.
(#valuations: 5673205, #canonical forms: 5673190, max #valuations
stored: 430758, #isomorphs removed: 5242530, #branches pruned: 0)

Generating Instance 65965 generated by valuation
S -> {S0}
T -> {T0}
Maj -> {(S0,T0)->{S0}}

Generating Instance 0 generated by valuation
S -> {S0}
T -> {T0}
Maj -> {(S0,T0)->{}}

Discarding Instance 0. The instance has no behaviour or is already covered by
smaller ones.

Generating Instance 131930 generated by valuation
S -> {S0,S1}
T -> {T0}
Maj -> {(S0,T0)->{},(S1,T0)->{}}

Generating Instance 1 generated by valuation
S -> {S0,S1,S2}
T -> {T0}
Maj -> {(S0,T0)->{},(S1,T0)->{S1,S2},(S2,T0)->{S1,S2}}

Instance 65965 written successfully to file raft_leader_election_instance_0.csp

Generating Instance 65966 generated by valuation
S -> {S0,S1,S2}
T -> {T0}
Maj -> {(S0,T0)->{S1,S2},(S1,T0)->{S1,S2},(S2,T0)->{S1,S2}}

Generating Instance 263860 generated by valuation
S -> {S0,S1}
T -> {T0}
Maj -> {(S0,T0)->{S0,S1},(S1,T0)->{S0,S1}}

Generating Instance 197895 generated by valuation
S -> {S0,S1}
T -> {T0}
Maj -> {(S0,T0)->{S0,S1},(S1,T0)->{}}

Instance 263860 written successfully to file raft_leader_election_instance_3.csp

.
.
.
```

```
Generating Instance 395788 generated by valuation
S -> {S0,S1,S2,S3,S4,S5,S6}
T -> {T0}
Maj -> {(S0,T0)->{S2,S3,S4,S5,S6},(S1,T0)->{S0,S1,S2,S3,S4,S5,S6},
(S2,T0)->{S0,S1,S2,S3,S4,S5,S6},(S3,T0)->{S0,S1,S2,S3,S4,S5,S6},
(S4,T0)->{S0,S1,S2,S3,S4,S5,S6},(S5,T0)->{S0,S2,S3,S4,S5,S6},
(S6,T0)->{S1,S2,S3,S4,S5,S6}}

Discarding Instance 395788. The instance has no behaviour or is already covered by
smaller ones.

Generating Instance 395789 generated by valuation
S -> {S0,S1,S2,S3,S4,S5,S6}
T -> {T0}
Maj -> {(S0,T0)->{S0,S1,S2,S3,S4,S5,S6},(S1,T0)->{S0,S1,S2,S3,S4,S5,S6},
(S2,T0)->{S0,S1,S2,S3,S4,S5,S6},(S3,T0)->{S0,S1,S2,S3,S4,S5,S6},
(S4,T0)->{S0,S1,S2,S3,S4,S5,S6},(S5,T0)->{S0,S1,S2,S3,S4,S5,S6},
(S6,T0)->{S0,S1,S2,S3,S4,S5,S6}}

Discarding Instance 395789. The instance has no behaviour or is already covered by
smaller ones.

Checking examples/raft_leader_election_instance_1.csp, this may take a while or two...

Checking examples/raft_leader_election_instance_2.csp, this may take a while or two...

Checking examples/raft_leader_election_instance_4.csp, this may take a while or two...

Checking examples/raft_leader_election_instance_3.csp, this may take a while or two...

Checking examples/raft_leader_election_instance_0.csp, this may take a while or two...

Checking examples/raft_leader_election_instance_5.csp, this may take a while or two...

Check of examples/raft_leader_election_instance_0.csp passed.

Checking examples/raft_leader_election_instance_6.csp, this may take a while or two...

Check of examples/raft_leader_election_instance_3.csp passed.

Checking examples/raft_leader_election_instance_9.csp, this may take a while or two...

.
.
.

Check of examples/raft_leader_election_instance_17.csp passed.

Check of examples/raft_leader_election_instance_14.csp passed.

Check of examples/raft_leader_election_instance_16.csp passed.

==== The system is correct with respect to the specification! ====

A total number of instances generated: 18
    (a total number of valuations generated: 5673205
     a total number of canonical forms computed: 5673190
     the maximum number of valuations stored all at once: 430758)
Total time taken: 311.084 seconds
    (time taken by input processing: 0 seconds
     time taken by the computation of valuations: 47.285 seconds
     time taken by output processing: 261.938 seconds
     time taken by trace refinement checking: 1.861 seconds)
```