An optimal cut-off algorithm for parameterised refinement checking

Siirtola, Antti

2020-10-15

Siirtola , A & Heljanko , K 2020 , ' An optimal cut-off algorithm for parameterised refinement checking ' , Science of Computer Programming , vol. 198 , 102517 . https://doi.org/10.1016/j.scico.2020.102517

http://hdl.handle.net/10138/346238
https://doi.org/10.1016/j.scico.2020.102517

An Optimal Cut-Off Algorithm for Parameterised Refinement Checking*

Antti Siirtola[a,*], Keijo Heljanko[b,c]

[a] *University of Oulu, Faculty of Information Technology and Electrical Engineering*
[b] *University of Helsinki, Department of Computer Science*
[c] *Helsinki Institute for Information Technology (HIIT)*

Abstract

The verification of contemporary distributed software systems is challenging, because they are heavily parameterised, containing components whose number and connections cannot be a priori fixed. In this work, we consider the multi-parameterised verification of safety properties by refinement checking in the context of labelled transition systems (LTSs). The LTSs are parameterised by using first-order constructs, sorts, variables, and predicates, while preserving compositionality. This allows us to parameterise not only the number of replicated components but also the communication topology of the system. Our approach to solving a verification task in the parameterised LTS formalism is to determine a finite cut-off set of parameter values such that in order to prove a parameterised system implementation correct with respect to its specification, it is sufficient to consider only finitely many instances of the parameterised system generated by the parameter values in the cut-off set.

In the conference version of this work, we converted the problem of determining a finite cut-off set into the unsatisfiability of a first-order formula and provided a satisfiability modulo theories (SMT)-based semi-algorithm for dynamically, i.e., iteratively, computing a cut-off set. In this article, we present a new version of the algorithm and prove that the cut-off sets computed by this new algorithm are optimal. Hence, we call the new version the optimal cut-off algorithm. The algorithm will always terminate for system topologies expressible in the $\exists^*\forall^*$ fragment of first-order logic. It also enables us to consider systems with topologies beyond this fragment, but for these systems, the algorithm is not guaranteed to terminate. We have implemented the approach on top of the Z3 SMT solver and successfully applied it to several system models. As a running example, we consider the leader election phase of the generalised (Byzantine) Raft consensus algorithm and prove the optimal cut-off set of six (respectively, thirteen) parameter values corresponding to instances up to three (respectively, four) servers. To the best of our knowledge, this is the first time a Byzantine variant of the parameterised Raft leader election is automatically verified.

Keywords: compositional verification, parameterized systems, cut-off, satisfiability modulo theories, automated verification

1. Introduction

We consider the compositional multi-parameterised verification of safety properties. The topic is theoretically interesting and practically relevant, because contemporary software systems are not only highly concurrent and distributed but also heavily parameterised containing components whose number and connections cannot be a priori fixed. Since these systems are everywhere around us, it is essential to verify

*This work is an extended and improved version of [1].

The final authenticated publication is available online at https://doi.org/10.1016/j.scico.2020.102517

*Corresponding author
Email addresses: antti.siirtola@oulu.fi (Antti Siirtola), keijo.heljanko@helsinki.fi (Keijo Heljanko)

that the critical ones operate properly in all circumstances, i.e., for all possible system parameter values. Moreover, since in large scale systems some subsystems (e.g., external software packages and subsystems concurrently under construction) can only be available in an interface specification form, we often need to be able to do their verification in a compositional way. Compositional verification is often preferable for scalability reasons, too.

Our Approach

Our formalism is based on the calculus of labelled transition systems (LTSs) with the trace refinement preorder and parallel composition and hiding operators [2, 3]. The LTSs are parameterised by using the constructs of first-order logic (FOL), *sorts* (a.k.a. *types*), typed *variables*, and *predicates*, such that compositional verification is possible in the parameterised setting, too. Sorts are used to parameterise the number of replicated components whereas predicates enable us to parameterise the system topology, i.e., the connections between the components.

Our goal is to solve a verification task in the parameterised LTS (PLTS) formalism by determining cut-offs for the parameters such that in order to prove a parameterised system implementation correct with respect to its specification, it is sufficient to consider only finitely many instances up to the cut-offs. In most other work, e.g. [4, 5, 6, 7, 8, 9, 10, 11, 12, 13], the only system parameters are integer variables that control the number of replicated components and, hence, cut-offs are simply integer values. In our case, sorts are used for the same purpose, to control the set of replicated components, and the cut-offs are integer values providing an upper bound to the size of sorts. However, in our formalism, we can also use predicates to parameterise the system topology. This makes the computation of cut-offs more complicated, but just like we can sometimes omit large sets of replicated components and only consider system instances with finitely many components, it is not always necessary to consider all possible connections between the components. In other words, it may be sufficient to limit our attention to finitely many values of predicates, i.e., the relations between the components. That is why we talk not only about the cut-offs (of sorts) but also about a cut-off set consisting of finitely many combinations of the values of sorts, predicates, and variables.

In the conference version of this work [1], we showed how the problem of determining a finite cut-off set can be converted into the unsatisfiability of a first-order formula and provided a satisfiability modulo theories (SMT)-based semi-algorithm for computing a cut-off set. The algorithm is called dynamic because it computes the cut-off set iteratively until the unsatisfiability condition is met. The downside of the algorithm is that it is not guaranteed to produce an optimal cut-off set and that its performance depends on the implementation of a heuristic oracle. Here, we overcome the limitations of the original algorithm and present a new version of the algorithm which does not involve an oracle and, upon termination, always produces the optimal cut-off set. Hence, we call the new version the optimal cut-off algorithm. The algorithm is implemented and successfully applied to several parameterised system models, including the repeatable read property of taDOM2+ XML database protocol [14] with the tree topology and the leader election phase of the generalised (Byzantine) Raft consensus protocol with the (Byzantine) quorum topology [15, 16].

Our approach is based on the precongruence reduction (PR) technique previously used to prove *static* cut-offs for PLTSs with predicates defined in the universal fragment ($\forall^*$) of FOL [17] and for PLTSs with special quorum functions [16]. The technique is also adapted to parameterised modal interface automata without predicates [18]. In general, the PR technique applies to parameterised systems with a finite basis, meaning that any (big) system instance can be represented as a composition of finitely many (small) system instances. For example, systems with a star, bipartite, totally connected, and quorum topology are such. However, systems with a linear, ring, or tree topology cannot be, in general, handled by our verification technique, because systems with such a topology can simulate a Turing machine [17]. On the other hand, if it is possible to capture the behaviour of the system from the viewpoint of any two components connected to each other in a PLTS, then one can study the transitive closures of rings, arrays and trees instead, since systems with such topologies have a finite basis. This is the modelling technique we have successfully applied, e.g., in the verification of the taDOM2+ protocol.

Static cut-off results [16, 17] are syntax-based and restricted to topologies specifiable in fragments of FOL. Consequently, a separate result is needed for each such fragment. The algorithms introduced here and in the conference version of this work [1] are not restricted to any syntactic fragment and they basically

allow us to utilise the full expressive power of FOL and treat the parameters of [16, 17] in a uniform way. Both algorithms will always terminate for topologies expressible in the $\exists^*\forall^*$ fragment of first-order logic, because the fragment is decidable [19] and systems with such topologies have a finite basis from which all instances can be composed [17]. The algorithms also enable us to consider systems with a finite basis beyond this fragment, but for these systems, termination depends on the capabilities of the used SMT solver.

Contribution

Compared with the conference version of this work [1], our main contribution is a new version of the dynamic cut-off algorithm which involves no heuristic oracle and which is now provably optimal in terms of the PR technique. This is because the cut-off set of a parameterised system is obtained by computing the minimal finite basis for the parameterised system. That is why the optimal cut-off algorithm introduced in this paper always produces at least as small or even smaller cut-off sets than the static cut-off methods and the original dynamic algorithm [1]. For example, the static cut-off for the number of servers in Raft is seven whereas in the computed optimal cut-off set the value is only three. Earlier, the cut-offs of 5-7 have been proved for other consensus algorithms [20]. We also provide the full proofs of the results, which were not included in [1].

The original dynamic algorithm [1] and the optimal cut-off algorithm of this paper also allow for the use of parameters that are beyond those handled by the static methods. As a new example, we consider the generalised version of the leader election of the Byzantine Raft [21], where some servers can be faulty and even malicious and therefore do not obey the protocol. The Byzantine Raft is out of the scope of static cut-off methods and infeasible for the original dynamic algorithm in [1]. However, it can be handled by the new optimal cut-off algorithm presented in this work and, consequently, we use the new algorithm to prove an optimal cut-off of four servers for the leader election phase of the Byzantine Raft. To the best of our knowledge, this is the first time a Byzantine variant of the parameterised Raft leader election is automatically verified.

In general, the distinctive features of our approach are compositionality, the support for multiple and topology related parameters, and the iterative computation of cut-offs. That is because many cut-off results only apply to systems with a single parameter determining the number of replicated components [5, 6, 7, 8, 9, 13].

Related Work

As regards compositionality, the process algebraic approaches of Valmari & Tienari [22], Lazić [23], and Creese [24] are the closest works. Valmari & Tienari [22] introduce a generic induction method for parameterised verification. Similar approaches are also presented by Kurshan & McMillan [25] and Wolper & Lovinfosse [26]. However, a crucial part of the technique is to come up with an invariant process which is a task that unfortunately cannot be automated in general. Lazić considers data-independent systems which can handle infinite or arbitrarily large data types. His approach allows the use of multiple parameters and provides static cut-offs for the size of data types. There is also a more general version of the results based on infinite automata [27], but in this context, compositionality is not considered. Moreover, neither approach [23, 27] allows the number of concurrent components nor the system topology to be parameterised. The limitation is overcome by Creese who combines the data-independence results with the induction method [24]. Simultaneously, however, full automation is lost.

Multi-parameterised verification was also considered by Emerson & Kahlon [4], Hanna et al. [28], and Yang & Li [11]. The approaches [4, 11] are based on static cut-offs, whereas [28] uses the iterative computation of cut-offs. The methods apply to systems with guarded broadcasts [4], shared actions [28], or rendezvous communication [11] and specifications are given in temporal logic [4, 28] or as property automata [11]. However, unlike in our work, the formalisms do not lend support to compositionality.

Clarke et al. [29] concern networks of homogeneous processes communicating through token passing. Their result applies to the systems of any topology but it allows for components of only one kind. Moreover, they only provide an upper bound for the size of network graphs but no algorithm for determining the networks up to the cut-off.

In addition to [22, 24, 28], iterative cut-off computation was previously considered by Kaiser, Kroening & Wahl [10], Abdulla, Haziza & Holik [12], and Liu & Wahl [30] as well. These approaches support the verification of reachability in the context of replicated Boolean programs [10], concurrent pushdown systems with a fixed number of threads parameterised by the number of context switches [30], and parameterised systems with various topologies (linear, multiset, ring, and tree) and communication primitives (global transitions, broadcasts, and rendezvous) [12]. However, neither technique lends support to multiple parameters nor compositional reasoning.

There are also several parameterised verification techniques based on abstract interpretation [31] and infinite-state verification algorithms on *well-structured transition systems (WSTSs)* [32]. While some of these techniques lend support to multiple parameters, they are typically not compositional. Moreover, it is unlikely that the PLTSs could be interpreted as WSTSs, because the PLTSs communicate through alphabet-based synchronisation, where increasing the number of replicated components may block some transitions and, hence, disable some behaviour. This easily breaks the key property of WSTSs which essentially requires that big systems have more behaviour than small ones. The additional benefit of our approach over these techniques is that we can exploit efficient finite-state model checkers for verification, and since abstraction is not involved, false negative verification results are avoided.

The Raft protocol is extensively verified earlier by using theorem proving [33]. However, this proof is Raft specific whereas our verification technique applies to a large class of parameterised systems and we only use the leader election phase of the (Byzantine) Raft as a running example in order to demonstrate the technique. Moreover, unlike in our work, Byzantine failures are not considered in [33].

Outline

The next three sections are preliminaries; they cover the basics of LTSs, FOL, and PLTSs. In Sect. 5, we present the PR technique for the verification of parameterised systems, and in Sect. 6, we introduce our main contribution, the optimal cut-off algorithm together with the proof of its correctness. The paper concludes with discussion on future research.

2. Labelled Transition Systems

In this section, we briefly recall a Communicating Sequential Processes (CSP)-like LTS-based process calculus with parallel composition and hiding operators and trace refinement preorder [3]. Basically, the only difference with the usual LTS notation is that events have an explicit data part which makes adding parameterisation convenient. The LTSs are used to express system components on both the implementation and specification side.

We assume a countably infinite set of *events*. One of the events is *invisible*, denoted τ, and the other ones 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 *atoms* and that each visible event is of the form $c(a_1, \ldots, a_n)$, where $c \in \mathbb{C}$ is a channel and $a_1, \ldots, a_n \in \mathbb{A}$ are atoms. Hence, an atom is the smallest piece of information that can be communicated via a channel.

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

Definition 2 (Parallel composition on LTSs). 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 $s_1' = s_1$.

Definition 3 (Hiding on LTSs). 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 $\alpha \in E'$.

For the purposes of verification, an LTS is interpreted as a set of finite traces. 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 is obtained from the execution by erasing all the states and the invisible events. The set of all the traces of L is denoted by $\mathrm{tr}(L)$.

Definition 4 (Trace refinement and equivalence). 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 $\mathrm{tr}(L_1) \subseteq \mathrm{tr}(L_2)$. 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 $L_2 \preceq_{\mathrm{tr}} L_1$ [2].

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. A system implementation is considered correct with respect to a system specification if the system implementation LTS is a trace refinement of the system specification LTS, i.e., every trace of the system implementation LTS is also a trace of the system specification LTS.

The operators and the trace relations have many useful properties [2, 3, 17], which are all exploited in the proofs. These properties are stated in Proposition 5, where $L_{id} := (\{\dot{s}\}, \emptyset, \emptyset, \dot{s})$ denotes a single-state LTS with the empty alphabet and no transition.

Proposition 5. *Let L_1, L_2, L_3 be LTSs and E a set of visible events. Then the following holds:*

1. $L_1 \parallel L_2 \equiv_{\mathrm{tr}} L_2 \parallel L_1$ *(commutativity),*
2. $L_1 \parallel (L_2 \parallel L_3) \equiv_{\mathrm{tr}} (L_1 \parallel L_2) \parallel L_3$ *(associativity),*
3. $L_1 \parallel L_1 \equiv_{\mathrm{tr}} L_1$ *(idempotence),*
4. $L_1 \parallel L_{id} \equiv_{\mathrm{tr}} L_1$ *(identity),*
5. $L_1 \setminus E = L_1 \setminus (E \cap \alpha(L_1))$ *(note that this is equality and, hence, stronger than trace equivalence),*
6. $(L_1 \parallel L_2) \setminus E \preceq_{\mathrm{tr}} (L_1 \setminus E) \parallel (L_2 \setminus E)$ *(partial distributivity),*
7. *if* $L_1 \preceq_{\mathrm{tr}} L_2$*, then* $L_1 \parallel L_3 \preceq_{\mathrm{tr}} L_2 \parallel L_3$ *and* $L_1 \setminus E \preceq_{\mathrm{tr}} L_2 \setminus E$ *(compositionality).*

The first four items state that the parallel composition is commutative, associative, and idempotent with respect to $\equiv_{\mathrm{tr}}$ and that L_{id} is the identity element of the parallel composition. This allows us to extend the parallel composition operator $\parallel$ to every finite set $I = \{i_1, \ldots, i_n\}$ and all LTSs $L_{i_1}, \ldots, L_{i_n}$ by defining

$$(\mathop{\Vert}_{i \in I} L_i) = (\mathop{\Vert}_{k=1}^{n} L_{i_k}) := \begin{cases} (L_{i_1} \parallel (\Vert_{i \in I \setminus \{i_1\}} L_i)), & \text{when } n > 0, \text{ and} \\ L_{id}, & \text{when } n = 0. \end{cases}$$

The last three items say that hiding events not in the alphabet of the LTS has no effect, distributing hiding over parallel composition results in an LTS greater in the preorder, and that $\preceq_{\mathrm{tr}}$ is compositional with respect to the parallel composition and hiding operators. Hence, $\preceq_{\mathrm{tr}}$ is a precongruence and $\equiv_{\mathrm{tr}}$ a congruence on LTSs.

Trace refinement allows us to consider safety but not liveness properties. That is because using a more expressive liveness-aware specification language, such as LTSs with failure-based semantics [3] or linear time temporal logic [34], easily breaks the partial distributivity of hiding over parallel composition (Item 6 of Proposition 5) and, in general, makes parameterised verification undecidable [17].

3. Many-Sorted First-Order Logic

In this section, we introduce first-order logic (FOL) [35] with *sorts* (a.k.a. *types*), typed *variables*, and *predicates*. We use FOL to express system topologies and quantifier-free formulae as guards in system descriptions.

We assume sets of sorts, variables, and predicates, denoted by $\mathbb{T}$, $\mathbb{X}$, and $\mathbb{F}$, respectively, that are disjoint and countably infinite. We assume that for each atom $a \in \mathbb{A}$, there is a sort $T_a \in \mathbb{T}$ and for each sort $T \in \mathbb{T}$, the set $\mathbb{A}_T := \{a \in \mathbb{A} \mid T_a = T\}$ is countably infinite. Hence, $\mathbb{A}_T$ and $\mathbb{A}_S$ are disjoint whenever T and S are different sorts. Moreover, we assume that for each variable $x \in \mathbb{X}$, there is a sort $T_x \in \mathbb{T}$, and for each predicate F, there is an arity $n_F \in \mathbb{N}$ and a tuple of sorts $\mathbf{T}_F = (T_F^1, \ldots, T_F^{n_F})$ specifying the domain of the predicate. We write $\mathbb{F}_n$ for the set of all predicates with the arity $n \in \mathbb{N}$.

Definition 6 (FOL). The *atomic formulae* of FOL are of the form $\top$ (always true), $x = y$ (equivalence), and $F(x_1, \ldots, x_n)$ (predicate application), where x and y are variables, F is a predicate with an arity n, and $x_1, \ldots, x_n$ are variables of the sort $T_F^1, \ldots, T_F^n$, respectively. The *formulae* of FOL are determined by the grammar

$$\mathcal{F} ::= p \mid \neg\mathcal{F} \mid (\mathcal{F} \wedge \mathcal{F}) \mid (\mathcal{F} \vee \mathcal{F}) \mid \forall x.\mathcal{F} \mid \exists x.\mathcal{F} \,,$$

where x denotes a variable and p an atomic formula.

We also write $x \neq y$ (inequivalence) and $\mathcal{F}_1 \to \mathcal{F}_2$ (implication) short for $\neg(x = y)$ and $(\neg\mathcal{F}_1) \vee \mathcal{F}_2$, respectively. A *quantifier-free formula*, also called a *guard*, is a formula without the quantified constructs of the form $\forall x.\mathcal{F}$ and $\exists x.\mathcal{F}$. A formula is in *negation normal form* if the negation $\neg$ is only applied to atomic formulae, i.e., no other connective or quantifier occurs under a negation. A formula is in the *prenex normal form* with *quantifier alternation* $Q_1, \ldots, Q_n$ if the formula is of the form $Q_1 x_1 . \cdots . Q_n x_n . \mathcal{F}$, where $Q_1, \ldots, Q_n \in \{\forall, \exists\}$ are quantifiers and $\mathcal{F}$ is quantifier-free.

Definition 7 (Signature of formula). The signature of a formula $\mathcal{F}$, denoted $\mathrm{sig}(\mathcal{F})$, is the set of the sorts, predicates, and free variables occurring in $\mathcal{F}$ and defined inductively as follows:

1. $\mathrm{sig}(\top) = \emptyset$,
2. $\mathrm{sig}(x = y) = \{x, y, T_x, T_y\}$,
3. $\mathrm{sig}(F(x_1, \ldots, x_n)) = \{F, x_1, \ldots, x_n, T_{x_1}, \ldots, T_{x_n}\}$,
4. $\mathrm{sig}(\neg\mathcal{F}) = \mathrm{sig}(\mathcal{F})$,
5. $\mathrm{sig}(\mathcal{F}_1 \wedge \mathcal{F}_2) = \mathrm{sig}(\mathcal{F}_1 \vee \mathcal{F}_2) = \mathrm{sig}(\mathcal{F}_1) \cup \mathrm{sig}(\mathcal{F}_2)$, and
6. $\mathrm{sig}(\forall x.\mathcal{F}) = \mathrm{sig}(\exists x.\mathcal{F}) = (\mathrm{sig}(\mathcal{F}) \setminus \{x\}) \cup \{T_x\}$ (x is considered bound).

The sorts, variables, and predicates in $\mathrm{sig}(\mathcal{F})$ are called the *parameters* of $\mathcal{F}$. We write $\mathrm{sig}_X(\mathcal{F})$ for the restriction $\mathrm{sig}(\mathcal{F}) \cap X$ of the signature to a set X.

A formula is evaluated to a Boolean value by using a *valuation* function which assigns values to sorts, variables, and predicates.

Definition 8 (Valuation). A *valuation* is a function ϕ such that

1. the domain of ϕ is a finite set of sorts, variables, and predicates,
2. for each sort $T \in \mathrm{dom}(\phi)$, $\phi(T)$ is a finite non-empty subset of $\mathbb{A}_T$,
3. for each variable $x \in \mathrm{dom}(\phi)$, $T_x \in \mathrm{dom}(\phi)$ and $\phi(x) \in \phi(T_x)$, and
4. for each predicate $F \in \mathrm{dom}(\phi)$, $T_F^1, \ldots, T_F^{n_F} \in \mathrm{dom}(\phi)$ and $\phi(F)$ is a subset of $\phi(T_F^1) \times \ldots \times \phi(T_F^{n_F})$.

We write $\mathrm{dom}_X(\phi)$ for the restriction $\mathrm{dom}(\phi) \cap X$ of the domain to a set X and $\mathrm{Im}(\phi)$ for the set $\bigcup_{T \in \mathrm{dom}_{\mathbb{T}}(\phi)} \phi(T)$ of all atoms in the image of ϕ. The complement $\overline{\phi(F)}$ of the value of a predicate F is the set $(\phi(T_F^1) \times \ldots \times \phi(T_F^{n_F})) \setminus \phi(F)$. A valuation ϕ is *compatible* with a formula $\mathcal{F}$ if $\mathrm{sig}(\mathcal{F}) \subseteq \mathrm{dom}(\phi)$.

A formula is evaluated to a Boolean value in the usual way by fixing the values of the parameters by using a compatible valuation ϕ and by evaluating the operators. Handling the propositional connectives is

straightforward but in order to treat quantification over a variable x, we need to consider all the extensions of a valuation to $\{x\}$. Let ϕ be a valuation and X a set of variables such that $T_x \in \mathrm{dom}_{\mathbb{T}}(\phi)$ for all $x \in X$. We write $\mathrm{ext}(\phi, X)$ for the set of all valuations ϕ' with the domain $\mathrm{dom}(\phi) \cup X$ such that $\phi'(x) \in \phi(T_x)$ for all $x \in X$ and $\phi'|_{\mathrm{dom}(\phi)\setminus X} = \phi|_{\mathrm{dom}(\phi)\setminus X}$, i.e., ϕ and ϕ' agree on the values of the parameters outside X.

Definition 9 (Instance of formula). Let $\mathcal{F}$ be a formula and ϕ a compatible valuation. The *ϕ-instance of $\mathcal{F}$* or the *instance of $\mathcal{F}$ (generated by ϕ)* is a Boolean value denoted by $[\![\mathcal{F}]\!]_\phi$ and evaluated inductively as follows:

1. $[\![\top]\!]_\phi = \mathit{true}$,
2. $[\![x = y]\!]_\phi = \begin{cases} \mathit{true}, & \text{if } \phi(x) = \phi(y), \\ \mathit{false}, & \text{if } \phi(x) \neq \phi(y), \end{cases}$
3. $[\![F(x_1, \ldots, x_n)]\!]_\phi = \begin{cases} \mathit{true}, & \text{if } (\phi(x_1), \ldots, \phi(x_n)) \in \phi(F), \\ \mathit{false}, & \text{if } (\phi(x_1), \ldots, \phi(x_n)) \notin \phi(F), \end{cases}$
4. $[\![\neg\mathcal{F}]\!]_\phi = \begin{cases} \mathit{true}, & \text{if } [\![\mathcal{F}]\!]_\phi = \mathit{false}, \\ \mathit{false}, & \text{if } [\![\mathcal{F}]\!]_\phi = \mathit{true}, \end{cases}$
5. $[\![(\mathcal{F}_1 \wedge \mathcal{F}_2)]\!]_\phi = \begin{cases} \mathit{true}, & \text{if } [\![\mathcal{F}_1]\!]_\phi = \mathit{true} \text{ and } [\![\mathcal{F}_2]\!]_\phi = \mathit{true}, \\ \mathit{false}, & \text{if } [\![\mathcal{F}_1]\!]_\phi = \mathit{false} \text{ or } [\![\mathcal{F}_2]\!]_\phi = \mathit{false}, \end{cases}$
6. $[\![(\mathcal{F}_1 \vee \mathcal{F}_2)]\!]_\phi = \begin{cases} \mathit{true}, & \text{if } [\![\mathcal{F}_1]\!]_\phi = \mathit{true} \text{ or } [\![\mathcal{F}_2]\!]_\phi = \mathit{true}, \\ \mathit{false}, & \text{if } [\![\mathcal{F}_1]\!]_\phi = \mathit{false} \text{ and } [\![\mathcal{F}_2]\!]_\phi = \mathit{false}, \end{cases}$
7. $[\![\forall x.\mathcal{F}]\!]_\phi = \begin{cases} \mathit{true}, & \text{if } [\![\mathcal{F}]\!]_{\phi'} = \mathit{true} \text{ for all } \phi' \in \mathrm{ext}(\phi, \{x\}), \\ \mathit{false}, & \text{if } [\![\mathcal{F}]\!]_{\phi'} = \mathit{false} \text{ for some } \phi' \in \mathrm{ext}(\phi, \{x\}), \text{ and} \end{cases}$
8. $[\![\exists x.\mathcal{F}]\!]_\phi = \begin{cases} \mathit{true}, & \text{if } [\![\mathcal{F}]\!]_{\phi'} = \mathit{true} \text{ for some } \phi' \in \mathrm{ext}(\phi, \{x\}), \\ \mathit{false}, & \text{if } [\![\mathcal{F}]\!]_{\phi'} = \mathit{false} \text{ for all } \phi' \in \mathrm{ext}(\phi, \{x\}). \end{cases}$

(Note that in Items 7 and 8, $\mathrm{ext}(\phi, \{x\})$ is finite and therefore, the ϕ-instance of $\mathcal{F}$ is computable.)

Formulae $\mathcal{F}_1$ and $\mathcal{F}_2$ are equivalent if they have the same signature and $[\![\mathcal{F}_1]\!]_\phi = [\![\mathcal{F}_2]\!]_\phi$ for every compatible valuation ϕ. Recall that for every formula there is an equivalent formula in the prenex normal form and for every (quantifier-free) formula there is an equivalent (respectively, quantifier-free) formula in negation normal form. Also note that the value of a predicate $F \in \mathbb{F}_0$ with the arity zero is either the empty set or the singleton set containing the empty tuple $()$. Therefore, F can be considered a Boolean variable b in the following sense: $\phi(b) = \mathit{true}$ if and only if $() \in \phi(F)$.

We say that a valuation ϕ *satisfies* a formula $\mathcal{F}$ or that $\mathcal{F}$ is *satisfiable* by ϕ, if ϕ is compatible with $\mathcal{F}$ and $[\![\mathcal{F}]\!]_\phi$ is *true*. The satisfiability problem in FOL can be formulated as follows: Given a formula $\mathcal{F}$, determine whether $\mathcal{F}$ is satisfiable by some valuation. Note that since our valuations are always finite, the satisfiability problem is equivalent to the question: Given a formula $\mathcal{F}$, determine a satisfying valuation of $\mathcal{F}$ or, if no such valuation exists, declare $\mathcal{F}$ unsatisfiable. The problem is undecidable in general, but the fragment consisting of the formulae with the quantifier alternation $\exists^*\forall^*$ is decidable [19, 36]. Many-sorted FOL considered here has several other known decidable fragments [36], too, but since many of them do not contain full $\exists^*\forall^*$, they are of limited use from the viewpoint of our algorithm introduced in Sect. 6.

4. Parameterised Labelled Transition Systems

In this section, we parameterise LTSs with first-order constructs, sorts, variables, and predicates, while preserving compositionality. Parameterised LTSs can express systems of various topologies with an unbounded number of replicated components and we use them to model both system implementations and specifications. As a running example, which is interleaved with the theory of PLTSs, we consider the leader

election phase of the Raft consensus algorithm [15] with and without Byzantine behaviour. With minor syntactic differences, the parameterisation of LTSs is done as in [1, 17], but the running example is revised from the conference version of this work [1] now being more general and covering also the Byzantine version of Raft. The machine readable version of the running example is included in Appendix A.

Example 10. In the Raft protocol [15], time is divided into *terms* of arbitrary length and a server can crash at any moment. When a server is running, it is in one of the 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 a vote request to the other servers and if it receives a quorum of votes, it becomes a leader. 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.

For our Raft model, we pick a sort T_S to represent the set $\{s_1, \ldots, s_n\} \subseteq \mathbb{A}_{T_S}$ of the identifiers of servers and a sort T_T to represent the set $\{t_1, \ldots, t_m\} \subseteq \mathbb{A}_{T_T}$ of the identifiers of terms. Variables x_i of the sort T_S are used to refer to individual servers and a variable y of the sort T_T is used to refer to a specific term.

Since a server can crash at any moment, the topology of a Raft system evolves over time. We model the changes in the topology by assigning each server x_0 and each term y a set $Q_{x_0,y}$ of servers from which the server x_0 may receive vote events. If the set $Q_{x_0,y}$ is empty the server x_0 cannot become a leader in the term y, otherwise the set $Q_{x_0,y}$ specifies the servers from which the server x_0 needs a vote in order to become a leader in the term y. The protocol assumes that the non-empty sets are *quorum sets*, i.e., they obey the *quorum property* [37]: any two quorum sets are overlapping.

In order to model this *quorum topology*, we use a predicate Q_S with $\mathbf{T}_{Q_S} = (T_S, T_T, T_S)$. For each server x_0 and each term y, we write $Q_{x_0,y} := \{x_1 \mid Q_S(x_0, y, x_1)\}$ for the set of servers from which x_0 may receive vote events in the term y. Now, in order to guarantee that the non-empty sets obey the quorum property, we require that for all servers x_0, x_1 and every term y, $Q_{x_0,y}$ is empty, $Q_{x_1,y}$ is empty, or the sets $Q_{x_0,y}$ and $Q_{x_1,y}$ are overlapping. Hence, the quorum topology is formally encoded as a formula

$$Qrm := \forall x_0.\forall x_1.\forall y.\big((\forall x_2.\neg Q_S(x_0, y, x_2)) \vee (\forall x_2.\neg Q_S(x_1, y, x_2)) \vee \exists x_2.(Q_S(x_0, y, x_2) \wedge Q_S(x_1, y, x_2))\big) \,.$$

The formula is outside the decidable fragment $\exists^*\forall^*$, because it involves a quantifier alternation $\forall\exists$ over the sort T_S, but our algorithm still terminates on this running example.

Note that in a practical implementation, the quorum property is guaranteed by requiring that a server may become a leader in a term only if it gets more than half the votes. That is because any two sets that cover more than half the base set are always overlapping. Since this is a special case of the quorum property, we call our model based on the quorum topology *Qrm* as the *generalised Raft*. □

Parameterised LTSs are constructed from LTSs (where variables are substituted for the atoms), quantifier-free formulae used as guards, and (replicated) parallel composition and hiding constructs which can be thought as operators on parameterised LTSs. Replicated parallel composition allows for parameterising the number of components while guards are used to restrict the system topology.

Definition 11 (PLTS). *Parameterised LTSs (PLTSs)* are defined inductively as follows:

1. If L is an LTS, $a_1, \ldots, a_n$ are the atoms occurring in $\alpha(L)$, and $x_1, \ldots, x_n$ are variables such that $T_{a_i} = T_{x_i}$ for all $i \in \{1, \ldots, n\}$, then a structure $L[x_1/a_1, \ldots, x_n/a_n]$ obtained from L by substituting x_i for each occurrence of a_i for each $i \in \{1, \ldots, n\}$ is an (elementary) PLTS.
2. If $\mathcal{P}$ is a PLTS and G a guard, then $([G]\,\mathcal{P})$ is a (guarded) PLTS.
3. If $\mathcal{P}_1$ and $\mathcal{P}_2$ are PLTSs, then $(\mathcal{P}_1 \parallel \mathcal{P}_2)$ is a (parallel) PLTS.
4. If $\mathcal{P}$ is a PLTS and x a variable, then $(\|_x\, \mathcal{P})$ is a (replicated parallel) PLTS.
5. If $\mathcal{P}$ is a PLTS and C a finite set of channels, $(\mathcal{P} \setminus C)$ is a (hiding) PLTS.

Definition 12 (Signature of PLTS). The signature function is extended to the set of PLTSs by setting:

1. $\mathrm{sig}(L[x_1/a_1, \ldots, x_n/a_n]) = \{x_1, \ldots, x_n, T_{x_1}, \ldots, T_{x_n}\}$,
2. $\mathrm{sig}([G]\,\mathcal{P}) = \mathrm{sig}(G) \cup \mathrm{sig}(\mathcal{P})$,
3. $\mathrm{sig}(\mathcal{P}_1 \parallel \mathcal{P}_2) = \mathrm{sig}(\mathcal{P}_1) \cup \mathrm{sig}(\mathcal{P}_2)$,
4. $\mathrm{sig}(\|_x \mathcal{P}) = (\mathrm{sig}(\mathcal{P}) \setminus \{x\}) \cup \{T_x\}$ (x is considered bound), and
5. $\mathrm{sig}(\mathcal{P} \setminus C) = \mathrm{sig}(\mathcal{P})$.

The signature determines the *parameters* of a PLTS and a valuation ϕ is said to be *compatible* with a PLTS $\mathcal{P}$ if $\mathrm{sig}(\mathcal{P}) \subseteq \mathrm{dom}(\phi)$. We sometimes write $\mathcal{P}(x_1, \ldots, x_n)$ to emphasise that $\mathcal{P}$ has the variables $x_1, \ldots, x_n$ as parameters.

A PLTS is evaluated to an LTS by fixing the values of the parameters by using a compatible valuation and by evaluating the operators.

Definition 13 (Instance of PLTS). Let $\mathcal{P}$ be a PLTS and ϕ a compatible valuation. The *ϕ-instance of $\mathcal{P}$* or the *instance of $\mathcal{P}$ (generated by ϕ)* is an LTS denoted by $[\![\mathcal{P}]\!]_\phi$ and evaluated inductively as follows:

1. $[\![L[x_1/a_1, \ldots, x_n/a_n]]\!]_\phi = L[\phi(x_1)/a_1, \ldots, \phi(x_n)/a_n]$,
2. $[\![[G]\,\mathcal{P}']\!]_\phi = \begin{cases} [\![\mathcal{P}']\!]_\phi, & \text{if } [\![G]\!]_\phi \text{ is } \mathit{true}, \\ L_{id}, & \text{if } [\![G]\!]_\phi \text{ is } \mathit{false} \text{ (the instance has no behaviour)}, \end{cases}$
3. $[\![\mathcal{P}_1 \parallel \mathcal{P}_2]\!]_\phi = [\![\mathcal{P}_1]\!]_\phi \parallel [\![\mathcal{P}_2]\!]_\phi$,
4. $[\![\|_x \mathcal{P}']\!]_\phi = \|_{\phi' \in \mathrm{ext}(\phi, \{x\})} [\![\mathcal{P}']\!]_{\phi'}$, and
5. $[\![\mathcal{P}' \setminus C]\!]_\phi = [\![\mathcal{P}']\!]_\phi \setminus \{c(a_1, \ldots, a_n) \in \alpha([\![\mathcal{P}']\!]_\phi) \mid c \in C\}$.

The successful application of the PLTS formalism necessitates a compositional modelling approach that is widely used in practice; a parameterised system implementation and specification are constructed from finitely many components by using parallel composition such that each component represents the behaviour of the system implementation or specification from the viewpoint of certain events and finitely many components. When modelling a system specification, this means it is sufficient that each illegal behaviour is forbidden by some system specification component.

Example 14. For the generalised Raft specification, we use an event $\mathit{leader}(x_0, y)$ to denote that the server x_0 is chosen as a leader in the term y. First, we consider the specification from the viewpoint of two servers, x_0 and x_1, and a term y. PLTS $\mathit{Spec2}(x_0, x_1, y)$ on the left of Figure 1 formally says that no two servers x_0 and x_1 can become a leader during the same term but repeating a leader notification is fine.

Recall the definition of the predicate Q_S from Example 10. As we let the variable y range over all term identifiers and x_0, x_1, and x_2 over all server identifiers, we obtain the model of the full specification as a PLTS

$$\mathit{Spec} := \|_{x_0} \|_{x_1} \|_{x_2} \|_{y} [Q_S(x_0, y, x_2) \wedge Q_S(x_1, y, x_2)]\, \mathit{Spec2}(x_0, x_1, y)\ ,$$

which says that for each term, there is at most one leader. The guard guarantees that for each term, we only consider servers with (non-empty) overlapping quorum sets. This is not a restriction since any two quorum sets are overlapping. Since $\mathit{Spec2}(x_0, x_1, y)$ has no bound variable, $\mathrm{sig}(\mathit{Spec2}(x_0, x_1, y)) = \{x_0, x_1, y, T_S, T_T\}$, but $\mathrm{sig}(\mathit{Spec}) = \{Q_S, T_S, T_T\}$ as *Spec* has only bound variables.

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 the sets $\{x \mid (s_i, t_1, x) \in \phi(Q_S)\}$, where $i \in \{1, \ldots, n\}$, obey the quorum property. Obviously, the valuation ϕ is compatible with *Spec* and the ϕ-instance of *Spec* is a star-shaped LTS on the right of Figure 1, which indeed says that there is at most one leader for the term t_1. □

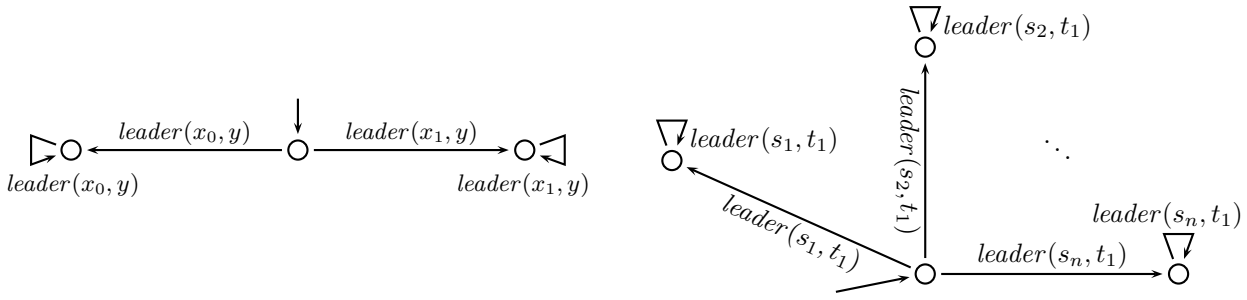

Figure 1: On the left: PLTS *Spec2*(x_0, x_1, y) representing the generalised Raft specification from the viewpoint of two servers, x_0 and x_1, and a term y. On the right: the instance of *Spec* representing the generalised Raft specification from the viewpoint of n servers $s_1, \ldots, s_n$ and a term t_1.

We complete the PLTS formalism by extending the trace refinement relation to the set of PLTSs while preserving compositionality. However, instead of a single relation, there will be infinitely many, since we use a formula to define the allowed values of parameters. This formula is basically just a system invariant which the valuations need to satisfy. Nevertheless, since we often use the formula to capture the system topology, we call it a *topology formula*, even though it can be used for specifying system invariants in general. Formally, a topology formula is a first-order formula accordant with Definition 6.

Definition 15 (Parameterised system/Refinement on PLTSs). A *parameterised system* is a triplet $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$, where $\mathcal{P}$ is an implementation PLTS, $\mathcal{Q}$ a specification PLTS, and $\mathcal{F}$ a topology formula. For a parameterised system $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$, we write $\mathcal{P} \preceq_{\mathrm{tr}}^{\mathcal{F}} \mathcal{Q}$ if and only if $[\![\mathcal{P}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\phi$ for all valuations ϕ that are compatible with $\mathcal{P}$ and $\mathcal{Q}$ and satisfy $\mathcal{F}$.

Obviously, Definition 15 can also be restricted to valuations with the minimal domain $\mathrm{sig}(\mathcal{P} \parallel \mathcal{Q}) \cup \mathrm{sig}(\mathcal{F})$. Therefore, we write $\mathrm{va}(\mathcal{F} \mid \mathcal{P}, \mathcal{Q})$ for the set of all valuations ϕ which have the domain $\mathrm{sig}(\mathcal{P} \parallel \mathcal{Q}) \cup \mathrm{sig}(\mathcal{F})$ and satisfy $\mathcal{F}$. We consider the parameterised system to be *correct* if $\mathcal{P} \preceq_{\mathrm{tr}}^{\mathcal{F}} \mathcal{Q}$. Since we use trace refinement, this allows for the verification of safety properties. Parameterised trace refinement relations also enable compositional verification since they are precongruences.

Proposition 16. *For all formulae $\mathcal{F}$, the relation $\preceq_{\mathrm{tr}}^{\mathcal{F}}$ is a precongruence on the set of PLTSs.*

Proof. The claim follows from Definition 13 and the precongruence of $\preceq_{\mathrm{tr}}$. The proof is similar to the proof of Proposition 17 in [17]. □

When the goal is to prove a system correct, it is safe to over-approximate the behaviour of the system implementation (components). This way, one can get false negative verification results, but if the model of the system implementation is correct with respect to the specification, also the original system is guaranteed to meet the specification.

Example 17. For the generalised Raft implementation, we use an 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 an 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 generalised Raft implementation is modelled in the same fashion as the specification. First, we capture it in the follower/candidate mode from the viewpoint of three servers x_0, x_1, x_2 and a term y in a PLTS *Flw3*(x_0, x_1, x_2, y) on the left of Figure 2. The PLTS says that in the term y, the server x_0 can vote for either x_1 or x_2, or become a candidate and vote for itself. When we let the variables x_1, x_2, and y 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/candidate mode as a PLTS

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

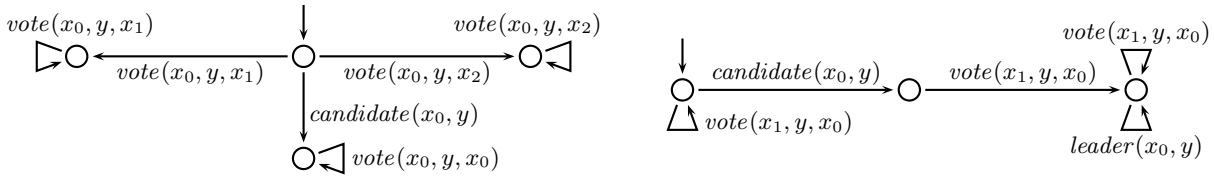

Figure 2: On the left: PLTS $\mathit{Flw3}(x_0, x_1, x_2, y)$ representing the generalised Raft implementation in the follower/candidate mode from the viewpoint of three servers x_0, x_1, x_2 and a term y. On the right: PLTS $\mathit{Ldr2}(x_0, x_1, y)$ representing the generalised Raft implementation in the candidate/leader mode from the viewpoint of two servers x_0, x_1 and a term y.

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

Second, we model the generalised Raft implementation in the candidate/leader mode from the viewpoint of two servers x_0, x_1 and a term y as a PLTS $\mathit{Ldr2}(x_0, x_1, y)$ on the right of Figure 2. 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 range over all term ids and x_1 range over all server ids in the set $Q_{x_0,y}$, the model of a single server x_0 running in the candidate/leader mode is obtained as a PLTS

$$\mathit{Ldr}(x_0) := \mathop{\|}_{y} \mathop{\|}_{x_1} [Q_S(x_0, y, x_1)]\, \mathit{Ldr2}(x_0, x_1, y) \ ,$$

which says that in order for a server to become a leader, it needs to become a candidate and then receive a vote from all the servers in $Q_{x_0,y}$.

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

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

Finally, we hide the events irrelevant to the specification yielding to a PLTS $\mathit{Raft}' := \mathit{Raft} \backslash \{\mathit{vote}, \mathit{candidate}\}$. Now, the problem on the correctness of the generalised Raft leader election can be formalised as the question whether

$$\mathit{Raft}' \preceq_{\mathrm{tr}}^{\mathit{Qrm}} \mathit{Spec}$$

holds. Code for the generalised Raft example is found in Appendix A.

Note that in our model, a quorum set of a server may change from term to term, but the changes in the topology are fixed a priori. This is not a restriction though as long as every possible combination of topology changes is covered. That is because we only consider safety properties, so it is sufficient that every trace of the system implementation is contained in some instance of the system model. □

Example 18. We also consider the Byzantine version [21] of the generalised Raft, where some malicious servers do not obey the protocol and therefore, may vote more than once in a term. In order to model the leader election phase of the generalised Byzantine Raft, we introduce a predicate NB with $\mathbf{T}_{\mathit{NB}} = (T_T, T_S)$ which specifies the set $\mathit{NB}_y := \{x \mid \mathit{NB}(y, x)\}$ of non-faulty servers for each term y. Hence, we consider a server x to operate correctly in the term y, if and only if $\mathit{NB}(y, x)$. In order to prove the correctness of generalised Byzantine Raft, we need to strengthen the quorum property to a condition which we call the *Byzantine quorum property*: for all x_0, x_1 and every term y, $Q_{x_0,y}$ is empty, $Q_{x_1,y}$ is empty, or the sets $Q_{x_0,y}$, $Q_{x_1,y}$, and NB_y are overlapping. Hence, the *Byzantine quorum topology* is formally encoded as a formula

$$\mathit{Byz} := \forall x_0.\forall x_1.\forall y.\big((\forall x_2.\neg Q_S(x_0, y, x_2)) \vee (\forall x_2.\neg Q_S(x_1, y, x_2)) \vee \exists x_2.(Q_S(x_0, y, x_2) \wedge Q_S(x_1, y, x_2) \wedge \mathit{NB}(y, x_2))\big) \ .$$

Also this formula is outside the decidable fragment $\exists^*\forall^*$, because it involves a quantifier alternation $\forall\exists$ over the sort T_S, but our algorithm still terminates on this running example, too.

Note that in a practical implementation, the Byzantine quorum property is guaranteed by requiring that in every term y, (i) more than two thirds of the servers are non-faulty and (ii) in order for a server to become a leader, it needs more than two thirds of the votes. Since the conditions (i) and (ii) imply the Byzantine quorum property, we call the Byzantine version of Raft with the Byzantine quorum topology, *Byz*, the *generalised Byzantine Raft*.

The correctness specification for the generalised Byzantine Raft is the same as for the generalised Raft. The implementation, however, is slightly different since the faulty servers may vote several times during a term. Hence, for the faulty servers, the PLTS $\mathit{Flw}(x_0, x_1, x_2, y)$ representing the correct behaviour in the follower/candidate mode is omitted. Therefore, the model of the generalised Byzantine Raft implementation becomes

$$\mathit{BRaft} := \|_{x_0} (\mathit{Ldr}(x_0) \|_{} (\|_{x_1\, x_2} \|[x_1 \neq x_2] \|_{y} [\mathit{NB}(y, x_0)]\, \mathit{Flw3}(x_0, x_1, x_2, y))) \; .$$

Finally, just like in the case of non-Byzantine generalised Raft, we hide the events irrelevant to the specification and formulate the problem on the correctness of the generalised Byzantine Raft leader election as the question whether

$$\mathit{BRaft} \setminus \{\mathit{vote}, \mathit{candidate}\} \preceq_{\mathrm{tr}}^{\mathit{Byz}} \mathit{Spec}$$

holds. Code for the generalised Byzantine Raft example is found in Appendix A. □

Our formalism enables the specification of various other topologies as well. For example, a classical ring topology can be modelled by using a sort N to represent the set of the nodes in a ring and a binary predicate C_N with $\mathbf{T}_{C_N} = (N, N)$ to represent the connections between them. The ring topology is then defined by the formula

$$\begin{aligned}\mathit{Rng} := (\forall z_0.\exists z_1.C_N(z_0, z_1)) \wedge (\forall z_0.\forall z_1.\forall z_2.((C_N(z_0, z_1) \wedge C_N(z_0, z_2)) \rightarrow z_1 = z_2)) \wedge \\ (\forall z_0.\exists z_1.C_N(z_1, z_0)) \wedge (\forall z_0.\forall z_1.\forall z_2.((C_N(z_1, z_0) \wedge C_N(z_2, z_0)) \rightarrow z_1 = z_2)) \; ,\end{aligned}$$

where z_0, z_1, z_2 are variables of the sort N, the first line states that each node has precisely one successor, and the last line says that each node has precisely one predecessor. If $\mathit{Adj2}(z_0, z_1)$ is an elementary PLTS capturing the behaviour of adjacent nodes z_0 and z_1, then the system with the ring topology can be represented as a PLTS

$$\|_{z_0\, z_1} \|[C_N(z_0, z_1)]\, \mathit{Adj2}(z_0, z_1) \; .$$

Actually, to be precise, *Rng* specifies a set of rings, but if this is undesirable and a single ring is preferred, we can introduce another binary predicate representing the transitive closure of *Rng* with one node removed and require that the transitive closure is irreflexive [17]. A system with a chain topology can be modelled in a similar fashion, we only need a specific head node and a specific tail node which do not have a predecessor and a successor, respectively.

Even though our formalism enables the specification of just about any topology, our algorithm does not terminate on all of them. For example, a system with the ring topology *Rng* does not have a finite basis, because larger rings cannot be constructed from smaller ones without breaking the existing ring structure. Moreover, there are PLTSs with a chain or ring topology that can simulate a Turing machine [17, 38], which implies that the verification of such systems is undecidable in general and it is only natural that our algorithm cannot handle them.

5. The Precongruence Reduction Technique

In this section, we present the precongruence reduction (PR) technique that enables us to reduce the question of the correctness of a parameterised system to finitely many refinement checks among LTSs [17]. The technique is based on determining a cut-off set although it does not directly give us an algorithm for computing one. Instead, it gives us the basic tools for reducing the number of needed refinement checks by utilising the algebraic properties of the refinement relation and the composition operators, especially

precongruence. As the main result of this section, we prove Proposition 31 which says that a finite basis of a parameterised system is a cut-off set. Results similar to Lemma 25 as well as Lemmata 26 and 29 are proved in [1] and [17, 18, 16], respectively, but here we also provide the full proofs which were not included in [1]. The main result, Proposition 31, is new.

Definition 19 (Cut-off set). Let $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$ be a parameterised system and $\Phi \subseteq \mathrm{va}(\mathcal{F} \mid \mathcal{P}, \mathcal{Q})$ a finite set of valuations. The set Φ is a *cut-off set* (for $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$), if $\mathcal{P} \preceq_{\mathrm{tr}}^{\mathcal{F}} \mathcal{Q}$ if and only if $[\![\mathcal{P}]\!]_\phi \preceq [\![\mathcal{Q}]\!]_\phi$ for all $\phi \in \Phi$.

Example 20. Recall our Raft example. In the next sections, we will show that in order to prove the generalised Raft correct, i.e., to solve the refinement $\mathit{Raft}' \preceq_{\mathrm{tr}}^{Qrm} \mathit{Spec}$, it is sufficient to consider the instances of the generalised Raft model up to three servers and a single term generated by the following six valuations:

1. $\phi_1(T_S) = \{s_1\}, \phi_1(T_T) = \{t_1\}, \phi_1(Q_S) = \{(s_1, t_1, s_1)\}$;
2. $\phi_2(T_S) = \{s_1, s_2\}, \phi_2(T_T) = \{t_1\}, \phi_2(Q_S) = \emptyset$;
3. $\phi_3(T_S) = \{s_1, s_2\}, \phi_3(T_T) = \{t_1\}, \phi_3(Q_S) = \{(s_1, t_1, s_2)\}$;
4. $\phi_4(T_S) = \{s_1, s_2\}, \phi_4(T_T) = \{t_1\}, \phi_4(Q_S) = \{(s_1, t_1, s_2), (s_2, t_1, s_2)\}$;
5. $\phi_5(T_S) = \{s_1, s_2, s_3\}, \phi_5(T_T) = \{t_1\}, \phi_5(Q_S) = \emptyset$; and
6. $\phi_6(T_S) = \{s_1, s_2, s_3\}, \phi_6(T_T) = \{t_1\}, \phi_6(Q_S) = \{(s_1, t_1, s_3), (s_2, t_1, s_3)\}$.

In other words, we will show that $\{\phi_1, \phi_2, \phi_3, \phi_4, \phi_5, \phi_6\}$ is a cut-off set for our generalised Raft model where the cut-offs are three and one for the size of T_S and T_T, respectively. We will also prove that the cut-off set is optimal with respect to the PR technique. □

The PR technique can be used to find cut-off sets for parameterised systems where the specification does not involve hiding. That is because hiding distributes only partially over parallel composition (Item 6 of Proposition 5) and consequently, the restriction is needed to prove Lemma 26. Moreover, allowing the use of hiding on the specification side makes the parameterised verification undecidable in general [17]. However, restricting the use of hiding is not a big limitation in practice since it 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 PR technique consists of the following steps. First, we show that if a big instance of the implementation PLTS $\mathcal{P}$ (respectively, a specification PLTS $\mathcal{Q}$) can be composed from a set of structurally smaller instances and each small instance of $\mathcal{P}$ is a trace refinement of the corresponding instance of $\mathcal{Q}$, then the big instance of $\mathcal{P}$ is a trace refinement of the big instance of $\mathcal{Q}$, too (Lemma 26). Second, we prove that valuations that can be obtained from each other by using the bijective renaming of atoms lead to equivalent verification tasks (Lemma 29). These two propositions together imply that if the system has a finite basis, i.e., all the instances are covered by a finite set of small instances, then the finite basis forms a cut-off set (Proposition 31).

In order to present the technique in detail, we first formalise the notion of the structure of an instance. Intuitively, the structure of an instance describes the full branches, from the root to a leaf, in the syntax tree of the instance. The branches are represented as finite strings, where each element in the string corresponds to an edge in the syntax tree.

Definition 21 (Structure of instance/Branch string). Let $\mathcal{P}$ be a PLTS and ϕ a compatible valuation. The *structure* (of the ϕ-instance of $\mathcal{P}$) is a finite set of finite strings denoted by $\mathrm{str}(\mathcal{P}, \phi)$ and defined inductively as follows:

1. $\mathrm{str}(L[x_1/a_1, \ldots, x_n/a_n], \phi) = \{\varepsilon\}$, where ε is the empty string,
2. $\mathrm{str}([G]\,\mathcal{P}', \phi) = \begin{cases} \mathrm{str}(\mathcal{P}', \phi), & \text{if } [\![G]\!]_\phi \text{ is } \mathit{true}, \\ \emptyset, & \text{otherwise,} \end{cases}$

3. $\mathrm{str}(\mathcal{P}_1 \parallel \mathcal{P}_2, \phi) = \{\mathit{false}\, w \mid w \in \mathrm{str}(\mathcal{P}_1, \phi)\} \cup \{\mathit{true}\, w \mid w \in \mathrm{str}(\mathcal{P}_2, \phi)\}$,
4. $\mathrm{str}(\|_x \mathcal{P}', \phi) = \bigcup_{\phi' \in \mathrm{ext}(\phi, \{x\})} \{\phi'(x)\, w \mid w \in \mathrm{str}(\mathcal{P}', \phi')\}$, and
5. $\mathrm{str}(\mathcal{P}' \setminus C, \phi) = \mathrm{str}(\mathcal{P}', \phi)$.

The elements of $\mathrm{str}(\mathcal{P}, \phi)$ are called *branch strings*.

Example 22. Recall our Raft model from Example 10. Let θ be a valuation in $\mathrm{va}(\mathit{Qrm} \mid \mathit{Raft}', \mathit{Spec})$ such that $\theta(T_S) = \{s_1, \ldots, s_n\}$, $\theta(T_T) = \{t_1, \ldots, t_m\}$, and for all $l \in \{1, \ldots, m\}$, the sets $\{x \mid (s_i, t_l, x) \in \theta(Q_S)\}$, where $i \in \{1, \ldots, n\}$, obey the quorum property. Then the structure of the parallel composition $\mathit{Raft}' \parallel \mathit{Spec}$ is the set

$$\mathrm{str}(\mathit{Raft}' \parallel \mathit{Spec}, \theta) = \Big(\bigcup_{i=1}^{n} \bigcup_{l=1}^{m} \bigcup_{\substack{j=1 \\ (s_i,t_l,s_j)\in\theta(Q_S)}}^{n} \{\mathit{false}\, s_i\, \mathit{false}\, t_l s_j\}\Big) \cup$$
$$\Big(\bigcup_{i=1}^{n} \bigcup_{j=1}^{n} \bigcup_{\substack{k=1 \\ j\neq k}}^{n} \bigcup_{l=1}^{m} \{\mathit{false}\, s_i\, \mathit{true}\, s_j s_k t_l\}\Big) \cup \Big(\bigcup_{i=1}^{n} \bigcup_{j=1}^{n} \bigcup_{k=1}^{n} \bigcup_{\substack{l=1 \\ (s_i,t_l,s_k),(s_j,t_l,s_k)\in\theta(Q_S)}}^{m} \{\mathit{true}\, s_i s_j s_k t_l\}\Big)\,. \quad \square$$

We say that the ϕ-instance of $\mathcal{P}$ is *smaller than (or equal to)* the ψ-instance of $\mathcal{P}$ if $\mathrm{str}(\mathcal{P}, \phi)$ is a subset of $\mathrm{str}(\mathcal{P}, \psi)$. Intuitively, smaller instances are generated by smaller valuations, called subvaluations, as stated in Lemma 25.

Definition 23 (Subvaluation). Let Υ be a set of sorts and Π, Ξ sets of predicates. A valuation ϕ is a (Υ, Π, Ξ)-*subvaluation* of a valuation ψ if and only if

1. the valuations have the same domain,
2. $\phi(T) \subseteq \psi(T)$ for all sorts $T \in \mathrm{dom}_{\mathbb{T}}(\phi) \cap \Upsilon$,
3. $\phi(T) = \psi(T)$ for all sorts $T \in \mathrm{dom}_{\mathbb{T}}(\phi) \setminus \Upsilon$,
4. $\phi(x) = \psi(x)$ for all variables $x \in \mathrm{dom}_{\mathbb{X}}(\phi)$,
5. $\phi(F) \subseteq \psi(F)$ for all predicates $F \in \mathrm{dom}_{\mathbb{F}}(\phi) \cap \Pi$, and
6. $\overline{\phi(F)} \subseteq \overline{\psi(F)}$ for all predicates $F \in \mathrm{dom}_{\mathbb{F}}(\phi) \cap \Xi$.

A (Υ, Π, Ξ)-subvaluation ϕ of ψ is *proper* if $\phi \neq \psi$, i.e., one of the containments above (2, 5, or 6) is proper.

The fact that ϕ is a (Υ, Π, Ξ)-*subvaluation* of ψ is denoted $\phi \subseteq_{\Upsilon,\Pi,\Xi} \psi$. Given a quantifier-free formula G, we write $\mathrm{pr}^+(G)$ and $\mathrm{pr}^-(G)$ for the set of predicates occurring under, respectively, an even and odd number of negations in G. We also write $\phi \subseteq_{\Upsilon,G} \psi$ or say that ϕ is a (Υ, G)-subvaluation of ψ to mean that ϕ is a $(\Upsilon, \mathrm{pr}^+(G), \mathrm{pr}^-(G))$-subvaluation of ψ. The notation is extended to PLTSs $\mathcal{P}$ by defining $\mathrm{pr}^{\oplus}(\mathcal{P})$, where $\oplus \in \{+, -\}$, as the union of all $\mathrm{pr}^{\oplus}(G)$ when G ranges over all guards in $\mathcal{P}$.

Example 24. Let θ be a valuation as in Example 22 and Θ the set of all valuations $\theta' \in \mathrm{va}(\mathit{Qrm} \mid \mathit{Raft}', \mathit{Spec})$ such that $\theta'(T_T) = \{t_l\}$, $\theta'(T_S) = \{s_i, s_j, s_k\}$, and $\theta'(Q_S) \subseteq \theta(Q_S)$ for some $l \in \{1, \ldots, m\}$ and $i, j, k \in \{1, \ldots, n\}$. Since $(\mathit{Raft}' \parallel \mathit{Spec})$ involves a single predicate, Q_S, without negation, $\mathrm{pr}^+(\mathit{Raft}' \parallel \mathit{Spec}) = \{Q_S\}$ and $\mathrm{pr}^-(\mathit{Raft}' \parallel \mathit{Spec}) = \emptyset$. This implies that Θ is a finite set of $(\mathbb{T}, \mathit{Raft}' \parallel \mathit{Spec})$-subvaluations of θ. It is also easy to see that for all $\theta' \in \Theta$, $\mathrm{str}(\mathit{Raft}' \parallel \mathit{Spec}, \theta')$ is a subset of $\mathrm{str}(\mathit{Raft}' \parallel \mathit{Spec}, \theta)$. $\square$

Lemma 25. *Let $\mathcal{P}$ be a PLTS, G a quantifier-free formula, and ψ, ϕ valuations compatible with $\mathcal{P}$ and G.*

1. *If ϕ is a $(\mathbb{T}, \emptyset, \emptyset)$-subvaluation of ψ and $\mathcal{P}$ is an elementary PLTS, then $[\![\mathcal{P}]\!]_\psi = [\![\mathcal{P}]\!]_\phi$ (note that this is equality and, hence, stronger than trace equivalence).*

2. *If ϕ is a $(\mathbb{T}, G)$-subvaluation of ψ and $[\![G]\!]_\phi$ is true, then $[\![G]\!]_\psi$ is true.*

3. *If ϕ is a $(\mathbb{T}, \mathcal{P})$-subvaluation of ψ, then* $\mathrm{str}(\mathcal{P}, \phi) \subseteq \mathrm{str}(\mathcal{P}, \psi)$.

Proof.

1. Since $\phi|_{\mathbb{X}} = \psi|_{\mathbb{X}}$ and an instance of an elementary PLTS is completely defined by the values of variables, the claim is evident.

2. First, recall that every quantifier-free formula G can be converted into an equivalent quantifier-free formula $\overline{G}$ which is in negation normal form. Moreover, this can be done in the standard way such that $\mathrm{pr}^+(G) = \mathrm{pr}^+(\overline{G})$ and $\mathrm{pr}^-(G) = \mathrm{pr}^-(\overline{G})$. Hence, without loss of generality, we may assume that G is in negation normal form. Then, we argue by induction on the structure of G by using the claim as the induction hypothesis.

   In the base case, there are five cases to consider because G cannot be $\neg\top$ which always evaluates to *false*. The case when G is $\top$ is obvious because $\top$ always evaluates to *true*. The cases when G is $x = y$ or $\neg x = y$ are easy, too. Since ϕ is a $(\mathbb{T}, G)$-subvaluation of ψ, $\phi(x) = \psi(x)$ and $\phi(y) = \psi(y)$, which implies that $[\![G]\!]_\phi = [\![G]\!]_\psi$. Let us then assume that G is $F(x_1, \ldots, x_n)$ and $[\![G]\!]_\phi$ is *true*. The former implies that $F \in \mathrm{pr}^+(G)$ and the latter means that $(\phi(x_1), \ldots, \phi(x_n)) \in \phi(F)$. Since ϕ is a $(\mathbb{T}, G)$-subvaluation of ψ, it implies that $(\psi(x_1), \ldots, \psi(x_n)) = (\phi(x_1), \ldots, \phi(x_n)) \in \phi(F) \subseteq \psi(F)$. Hence, $[\![G]\!]_\psi$ is *true* as well. The case when G is $\neg F(x_1, \ldots, x_n)$ is similar.

   In the induction step, there are two cases two consider. First, let us assume that G is $G_1 \wedge G_2$ and $[\![G]\!]_\phi$ is *true*, i.e., $[\![G_1]\!]_\phi$ and $[\![G_2]\!]_\phi$ are *true*. Obviously, for both $i \in \{1, 2\}$, ϕ is a $(\mathbb{T}, G_i)$-subvaluation of ψ, which by the induction hypothesis, implies that $[\![G_i]\!]_\psi$ is *true*. Hence, $[\![G]\!]_\psi$ is *true* as well. Second, if G is $G_1 \vee G_2$ and $[\![G]\!]_\phi$ is *true*, then $[\![G_i]\!]_\phi$ is *true* for some $i \in \{1, 2\}$. Since ϕ is a $(\mathbb{T}, G_i)$-subvaluation of ψ, by the induction hypothesis, it implies that $[\![G_i]\!]_\psi$ is *true* and, hence, $[\![G]\!]_\psi$ is *true* as well.

   Therefore, by the induction principle, the second claim of the lemma holds.

3. We argue by induction on the structure of $\mathcal{P}$ by using the lemma as the induction hypothesis.

   In the base case, $\mathcal{P}$ is an elementary PLTS, which implies that $\mathrm{str}(\mathcal{P}, \phi) = \mathrm{str}(\mathcal{P}, \psi) = \{\varepsilon\}$ and the claim is obvious.

   In the induction step, there are four cases to consider. First, let us assume that $\mathcal{P}$ is $([G]\,\mathcal{P}')$. Then ϕ is a $(\mathbb{T}, G)$- and $(\mathbb{T}, \mathcal{P}')$-subvaluation of ψ. If $[\![G]\!]_\phi$ is *false*, then $\mathrm{str}(\mathcal{P}, \phi)$ is the empty set and obviously contained in $\mathrm{str}(\mathcal{P}, \psi)$. On the other hand, if $[\![G]\!]_\phi$ is *true*, then by the second claim of the lemma, $[\![G]\!]_\psi$ is *true* as well. Since ϕ is a $(\mathbb{T}, \mathcal{P}')$-subvaluation of ψ, by Definition 21 and the induction hypothesis, it implies that $\mathrm{str}(\mathcal{P}, \phi) = \mathrm{str}(\mathcal{P}', \phi) \subseteq \mathrm{str}(\mathcal{P}', \psi) = \mathrm{str}(\mathcal{P}, \psi)$.

   Second, if $\mathcal{P}$ is $\mathcal{P}_1 \parallel \mathcal{P}_2$, then ϕ is a $(\mathbb{T}, \mathcal{P}_i)$-subvaluation of ψ for both $i \in \{1, 2\}$. By the induction hypothesis, it implies that $\mathrm{str}(\mathcal{P}_i, \phi) \subseteq \mathrm{str}(\mathcal{P}_i, \psi)$ for both $i \in \{1, 2\}$, and by Definition 21, we see that $\mathrm{str}(\mathcal{P}, \phi) \subseteq \mathrm{str}(\mathcal{P}, \psi)$.

   Third, we assume that $\mathcal{P}$ is $\|_x \mathcal{P}'$. Since ϕ is a $(\mathbb{T}, \mathcal{P})$-subvaluation of ψ, for every $\phi' \in \mathrm{ext}(\phi, \{x\})$, there is $\psi' \in \mathrm{ext}(\psi, \{x\})$ such that ϕ' is a $(\mathbb{T}, \mathcal{P}')$-subvaluation of ψ'. Hence, by the induction hypothesis, $\mathrm{str}(\mathcal{P}', \phi') \subseteq \mathrm{str}(\mathcal{P}', \psi')$ for every such ϕ' and ψ'. By Definition 21, this implies that $\mathrm{str}(\mathcal{P}, \phi) \subseteq \mathrm{str}(\mathcal{P}, \psi)$.

   Finally, if $\mathcal{P}$ is $\mathcal{P}' \setminus C$, then ϕ is a $(\mathbb{T}, \mathcal{P}')$-subvaluation of ψ. By Definition 21 and the induction hypothesis, this implies that $\mathrm{str}(\mathcal{P}, \phi) = \mathrm{str}(\mathcal{P}', \phi) \subseteq \mathrm{str}(\mathcal{P}', \psi) = \mathrm{str}(\mathcal{P}, \psi)$.

   Therefore, by the induction principle, also the third claim of the lemma holds. □

With the aid of the lemma above, we can show that the correctness of a (big) implementation instance can be derived from the correctness of structurally smaller instances if the big implementation and specification instances are composed of the structurally smaller, respectively, implementation and specification instances.

Lemma 26. *Let $\mathcal{P}$ be an implementation PLTS, $\mathcal{Q}$ a specification PLTS, ψ a valuation compatible with $\mathcal{P}$ and $\mathcal{Q}$, and Φ a finite set of $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluations of ψ. If* $\mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, \psi) = \bigcup_{\phi \in \Phi} \mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, \phi)$ *and $[\![\mathcal{P}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\phi$ for all $\phi \in \Phi$, then $[\![\mathcal{P}]\!]_\psi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\psi$.*

PROOF. First, we argue that if Φ is a finite set of $(\mathbb{T},\mathcal{P})$-subvaluations of ψ and $\mathrm{str}(\mathcal{P},\psi)=\bigcup_{\phi\in\Phi}\mathrm{str}(\mathcal{P},\phi)$, then $[\![\mathcal{P}]\!]_\psi \preceq_{\mathrm{tr}} \|_{\phi\in\Phi}[\![\mathcal{P}]\!]_\phi$. The proof proceeds by induction on the structure of $\mathcal{P}$ by using the claim as the induction hypothesis.

In the base case, $\mathcal{P}$ is an elementary PLTS. Since ϕ is a $(\mathbb{T},\emptyset,\emptyset)$-subvaluation of ψ for all $\phi\in\Phi$, by Item 1 of Lemma 25, we know that $[\![\mathcal{P}]\!]_\psi=[\![\mathcal{P}]\!]_\phi$. Since every LTS is idempotent with respect to parallel composition (Proposition 5, Item 3) and $\preceq_{\mathrm{tr}}$ is transitive, it implies that $[\![\mathcal{P}]\!]_\psi \preceq_{\mathrm{tr}} \|_{\phi\in\Phi}[\![\mathcal{P}]\!]_\phi$.

In the induction step, we have four cases to consider. Let us first assume that $\mathcal{P}$ is $([G]\,\mathcal{P}')$. If $[\![G]\!]_\psi$ is *false*, then by the second item of Lemma 25, $[\![G]\!]_\phi$ is *false* for all $\phi\in\Phi$. This implies that $[\![\mathcal{P}]\!]_\psi = L_{id} \equiv_{\mathrm{tr}} \|_{\phi\in\Phi}[\![\mathcal{P}]\!]_\phi$. Let us then assume that $[\![G]\!]_\psi$ is *true* and let Φ_t be the set of all $\phi\in\Phi$ such that $[\![G]\!]_\phi$ is *true*. Then Φ_t is a finite set of $(\mathbb{T},\mathcal{P}')$-subvaluations of ψ. Moreover, since $\mathrm{str}(\mathcal{P},\phi)$ is empty for all $\phi\in\Phi\setminus\Phi_t$, it means that $\mathrm{str}(\mathcal{P}',\psi)=\mathrm{str}(\mathcal{P},\psi)=\bigcup_{\phi\in\Phi}\mathrm{str}(\mathcal{P},\phi)=\bigcup_{\phi\in\Phi_t}\mathrm{str}(\mathcal{P}',\phi)$. By the induction hypothesis and the identity of L_{id} (Proposition 5, Item 4), it implies that

$$[\![\mathcal{P}]\!]_\psi = [\![\mathcal{P}']\!]_\psi \overset{\text{i.h.}}{\preceq_{\mathrm{tr}}} \|_{\phi\in\Phi_t}[\![\mathcal{P}']\!]_\phi \overset{\text{P5.4}}{\equiv_{\mathrm{tr}}} \|_{\phi\in\Phi}[\![\mathcal{P}]\!]_\phi\ .$$

If $\mathcal{P}$ is $\mathcal{P}_1\parallel\mathcal{P}_2$, then, by the assumption, we know that Φ is a finite set of $(\mathbb{T},\mathcal{P}_i)$-subvaluations of ψ and $\mathrm{str}(\mathcal{P}_i,\psi)=\bigcup_{\phi\in\Phi}\mathrm{str}(\mathcal{P}_i,\phi)$ for both $i\in\{1,2\}$. Then, by the induction hypothesis, we see that $[\![\mathcal{P}_i]\!]_\psi \preceq_{\mathrm{tr}} \|_{\phi\in\Phi}[\![\mathcal{P}_i]\!]_\phi$ for both $i\in\{1,2\}$. By the commutativity and associativity of the parallel composition (Proposition 5, Items 1–2) and the compositionality of the trace refinement (Proposition 5, Item 7), we have

$$[\![\mathcal{P}]\!]_\psi = [\![\mathcal{P}_1]\!]_\psi \parallel [\![\mathcal{P}_2]\!]_\psi \overset{\text{i.h. \& P5.7}}{\preceq_{\mathrm{tr}}} (\|_{\phi\in\Phi}[\![\mathcal{P}_1]\!]_\phi) \parallel (\|_{\phi\in\Phi}[\![\mathcal{P}_2]\!]_\phi) \overset{\text{P5.1–2}}{\equiv_{\mathrm{tr}}} \|_{\phi\in\Phi}([\![\mathcal{P}_1]\!]_\phi \parallel [\![\mathcal{P}_2]\!]_\phi) = \|_{\phi\in\Phi}[\![\mathcal{P}]\!]_\phi\ .$$

By the transitivity of the trace refinement, it implies that $[\![\mathcal{P}]\!]_\psi \preceq_{\mathrm{tr}} \|_{\phi\in\Phi}[\![\mathcal{P}]\!]_\phi$.

Next, we assume that $\mathcal{P}$ is $\|_x\mathcal{P}'$. For every $a\in\psi(T_x)$, let Φ_a denote the set of all valuations $\phi'\in\bigcup_{\phi\in\Phi}\mathrm{ext}(\phi,\{x\})$ such that $\phi'(x)=a$. Obviously, $\Phi_{\psi'(x)}$ is a finite set of $(\mathbb{T},\mathcal{P}')$-subvaluations of ψ' for every $\psi'\in\mathrm{ext}(\psi,\{x\})$. Moreover, since $\mathrm{str}(\mathcal{P},\psi)=\bigcup_{\phi\in\Phi}\mathrm{str}(\mathcal{P},\phi)$, then $\mathrm{str}(\mathcal{P},\psi')$ must be equal to $\bigcup_{\phi'\in\Phi_{\psi'(x)}}\mathrm{str}(\mathcal{P},\phi')$ for all $\psi'\in\mathrm{ext}(\psi,\{x\})$. By the induction hypothesis, it implies that $[\![\mathcal{P}']\!]_{\psi'} \preceq_{\mathrm{tr}} \|_{\phi'\in\Phi_{\psi'(x)}}[\![\mathcal{P}']\!]_{\phi'}$ for every $\psi'\in\mathrm{ext}(\psi,\{x\})$. Then, by the commutativity, associativity, idempotence, and identity of the parallel composition (Proposition 5, Items 1–4) and the compositionality of the trace refinement (Proposition 5, Item 7), the following holds:

$$\begin{aligned}[\![\mathcal{P}]\!]_\psi = \|_{\psi'\in\mathrm{ext}(\psi,\{x\})}[\![\mathcal{P}']\!]_{\psi'} &\overset{\text{i.h. \& P5.7}}{\preceq_{\mathrm{tr}}} \|_{\psi'\in\mathrm{ext}(\psi,\{x\})}(\|_{\phi'\in\Phi_{\psi'(x)}}[\![\mathcal{P}']\!]_{\phi'}) \\ &\overset{\text{P5.1–4}}{\equiv_{\mathrm{tr}}} \|_{\phi'\in\bigcup_{\phi\in\Phi}\mathrm{ext}(\phi,\{x\})}[\![\mathcal{P}']\!]_{\phi'} \overset{\text{P5.1–3}}{\equiv_{\mathrm{tr}}} \|_{\phi\in\Phi}(\|_{\phi'\in\mathrm{ext}(\phi,\{x\})}[\![\mathcal{P}']\!]_{\phi'}) = \|_{\phi\in\Phi}[\![\mathcal{P}]\!]_\phi\ .\end{aligned}$$

By the transitivity of the trace refinement, it implies that $[\![\mathcal{P}]\!]_\psi \preceq_{\mathrm{tr}} \|_{\phi\in\Phi}[\![\mathcal{P}]\!]_\phi$.

Finally, if $\mathcal{P}$ is $\mathcal{P}'\setminus C$, then we write E for the set $\{c(a_1,\ldots,a_n)\mid c\in C, a_1,\ldots,a_n\in\mathbb{A}\}$ of all events that can be communicated via the channels in C. By the induction hypothesis, we learn that $[\![\mathcal{P}']\!]_\psi \preceq_{\mathrm{tr}} \|_{\phi\in\Phi}[\![\mathcal{P}']\!]_\phi$, which implies that $\alpha([\![\mathcal{P}']\!]_\phi)\subseteq\alpha([\![\mathcal{P}']\!]_\psi)$ for all $\phi\in\Phi$. By the properties of hiding (Proposition 5, Items 5–6) and the compositionality of the trace refinement (Proposition 5, Item 7), this implies that

$$\begin{aligned}[\![\mathcal{P}]\!]_\psi = [\![\mathcal{P}']\!]_\psi \setminus (E\cap\alpha([\![\mathcal{P}']\!]_\psi)) \overset{\text{P5.5}}{=} [\![\mathcal{P}']\!]_\psi\setminus E &\overset{\text{i.h. \& P5.7}}{\preceq_{\mathrm{tr}}} (\|_{\phi\in\Phi}[\![\mathcal{P}']\!]_\phi)\setminus E \overset{\text{P5.6–7}}{\preceq_{\mathrm{tr}}} \\ &\|_{\phi\in\Phi}([\![\mathcal{P}']\!]_\phi\setminus E) \overset{\text{P5.5}}{=} \|_{\phi\in\Phi}([\![\mathcal{P}']\!]_\phi\setminus(E\cap\alpha([\![\mathcal{P}']\!]_\phi))) = \|_{\phi\in\Phi}[\![\mathcal{P}]\!]_\phi\ .\end{aligned}$$

By the transitivity of the trace refinement, it implies that $[\![\mathcal{P}]\!]_\psi \preceq_{\mathrm{tr}} \|_{\phi\in\Phi}[\![\mathcal{P}]\!]_\phi$.

The proof that Φ is a finite set of $(\mathbb{T},\mathcal{Q})$-subvaluations of ψ and $\mathrm{str}(\mathcal{Q},\psi)=\bigcup_{\phi\in\Phi}\mathrm{str}(\mathcal{Q},\phi)$ implies $\|_{\phi\in\Phi}[\![\mathcal{Q}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\psi$ is similar but simpler because there is no need to consider the hiding construct.

If Φ is a finite set of $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluations of ψ, $\mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, \psi) = \bigcup_{\phi \in \Phi} \mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, \phi)$, and $[\![\mathcal{P}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\phi$ for all $\phi \in \Phi$, then by above and the precongruence of $\preceq_{\mathrm{tr}}$, $[\![\mathcal{P}]\!]_\psi \preceq_{\mathrm{tr}} \|_{\phi \in \Phi} [\![\mathcal{P}]\!]_\phi \preceq_{\mathrm{tr}} \|_{\phi \in \Phi} [\![\mathcal{Q}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\psi$. □

Example 27. Let θ and Θ be as in Examples 22 and 24. Since every branch string in $\mathrm{str}(\mathit{Raft}' \parallel \mathit{Spec}, \theta)$ depends on the identifiers of at most three servers and one term, it is easy to see that $\mathrm{str}(\mathit{Raft}' \parallel \mathit{Spec}, \theta) = \bigcup_{\theta' \in \Theta} \mathrm{str}(\mathit{Raft}' \parallel \mathit{Spec}, \theta')$, i.e., the structure of the θ-instance of $(\mathit{Raft}' \parallel \mathit{Spec})$ consists of the same branch strings as the structure of θ'-instances, where $\theta' \in \Theta$. Since Θ is finite, by Lemma 26, 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. Note, however, that not all valuations in Θ satisfy Qrm, so this does not yet mean that the correctness of our Raft model can be derived from the correctness of its instances up to three servers and one term. □

Valuations that can be obtained from each other by the bijective renaming of atoms result in equivalent verification tasks. A function (injection) $g : A \to B$, where $A, B \subseteq \mathbb{A}$, is a *sortwise function* (respectively, *injection*) if $g(a) \in \mathbb{A}_{T_a}$ for each $a \in A$. Let ϕ be a valuation and g a sortwise function: $\mathrm{Im}(\phi) \to \mathbb{A}$. We write $g(\phi)$ for a valuation ϕ' which is obtained from ϕ by mapping the atoms in the image using g.

Definition 28 (Isomorphism on valuations). Valuations ϕ_1 and ϕ_2 are *isomorphic*, if there is a sortwise injection $g : \mathrm{Im}(\phi_1) \to \mathrm{Im}(\phi_2)$ such that $\phi_2 = g(\phi_1)$.

Lemma 29. *Let $\mathcal{V}$ be a formula, $\mathcal{P}$ and $\mathcal{Q}$ PLTSs, and ϕ_1 and ϕ_2 isomorphic valuations.*

1. *If ϕ_1 and ϕ_2 are compatible with $\mathcal{V}$, then $[\![\mathcal{V}]\!]_{\phi_1}$ if and only if $[\![\mathcal{V}]\!]_{\phi_2}$.*
2. *If ϕ_1 and ϕ_2 are compatible with $\mathcal{P}$ and $\mathcal{Q}$, then $[\![\mathcal{P}]\!]_{\phi_1} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\phi_1}$ if and only if $[\![\mathcal{P}]\!]_{\phi_2} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\phi_2}$.*

Proof. The first is claim is easy to prove by induction on the structure of $\mathcal{V}$.

In order to prove the second claim, we define renaming on LTSs [3, 2]. If L is an LTS, A_L the set of all atoms occurring in L, and g a sortwise injection: $A_L \to \mathbb{A}$, then $g(L)$ denotes an LTS obtained from L by substituting $g(a)$ for every occurrence of every atom $a \in A_L$. Obviously, renaming is a compositional operator on LTSs, i.e., $L_1 \preceq_{\mathrm{tr}} L_2$ implies $g(L_1) \preceq_{\mathrm{tr}} g(L_2)$ for all LTSs L_1, L_2 and every sortwise injection g from $A_{L_1} \cup A_{L_2}$, and renaming distributes over the parallel composition and hiding operators, i.e., $g(L_1 \parallel L_2) \equiv_{\mathrm{tr}} g(L_1) \parallel g(L_2)$ and $g(L_1 \setminus E) \equiv_{\mathrm{tr}} g(L_1) \setminus g(E)$ for all LTSs L_1, L_2, every set E of visible events, and every sortwise injection g from $A_{L_1} \cup A_{L_2} \cup E$.

Now, by utilising the properties above, we can show that if $\mathcal{P}$ is a PLTS, ϕ a compatible valuation, and g a sortwise injection: $\mathrm{Im}(\phi) \to \mathbb{A}$, then $g([\![\mathcal{P}]\!]_\phi) = [\![\mathcal{P}]\!]_{g(\phi)}$. This is easy to prove by induction on the structure of $\mathcal{P}$.

Finally, since ϕ_1 and ϕ_2 are isomorphic valuations, there is a sortwise injection $g : \mathrm{Im}(\phi_1) \to \mathrm{Im}(\phi_2)$ such that $\phi_2 = g(\phi_1)$. Now, if $[\![\mathcal{P}]\!]_{\phi_1} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\phi_1}$, then by the compositionality of renaming, $g([\![\mathcal{P}]\!]_{\phi_1}) \preceq_{\mathrm{tr}} g([\![\mathcal{Q}]\!]_{\phi_1})$, too. By above, it means that $[\![\mathcal{P}]\!]_{g(\phi_1)} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{g(\phi_1)}$ or, in other words, $[\![\mathcal{P}]\!]_{\phi_2} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\phi_2}$. Similarly, we can prove that $[\![\mathcal{P}]\!]_{\phi_2} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\phi_2}$ implies $[\![\mathcal{P}]\!]_{\phi_1} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\phi_1}$. Hence, $[\![\mathcal{P}]\!]_{\phi_1} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\phi_1}$ if and only if $[\![\mathcal{P}]\!]_{\phi_2} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{\phi_2}$. □

Lemmata 26 and 29 imply that if the system has a finite basis in the sense that all the instances are covered by a finite set of (injectively renamed) small instances, then we can reduce a refinement checking on PLTSs to finitely many refinement checks on LTSs. This is stated formally as Proposition 31.

Definition 30 (Basis). Let $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$ be a parameterised system. A set $\Phi \subseteq \mathrm{va}(\mathcal{F} \mid \mathcal{P}, \mathcal{Q})$ of valuations is a *basis* (of $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$) if for all $\psi \in \mathrm{va}(\mathcal{F} \mid \mathcal{P}, \mathcal{Q})$ and all $w \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, \psi)$, there is a valuation $\phi \in \Phi$ and a sortwise injection $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\psi)$ such that $w \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, g(\phi))$ and $g(\phi)$ is a $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of ψ.

Proposition 31. *A finite basis of a parameterised system $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$ is a cut-off set for $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$.*

PROOF. In order to prove that Φ is a cut-off set for $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$, it is sufficient to show that if $[\![\mathcal{P}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\phi$ for all $\phi \in \Phi$, then $[\![\mathcal{P}]\!]_\psi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\psi$ for all $\psi \in \mathrm{va}(\mathcal{F} \mid \mathcal{P}, \mathcal{Q})$, because the opposite implication is trivial. Hence, let us assume that Φ is a finite basis, $[\![\mathcal{P}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\phi$ for all $\phi \in \Phi$, and $\psi \in \mathrm{va}(\mathcal{F} \mid \mathcal{P}, \mathcal{Q})$. Since Φ is a basis, for every $w \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, \psi)$, there is a valuation $\phi_w \in \Phi$ and a sortwise injection $g_w : \mathrm{Im}(\phi_w) \to \mathrm{Im}(\psi)$ such that $w \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, g_w(\phi_w))$ and $g_w(\phi_w)$ is a $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of ψ. By Item 3 of Lemma 25, $[\![\mathcal{P} \parallel \mathcal{Q}]\!]_{g_w(\phi_w)}$ is smaller than $[\![\mathcal{P} \parallel \mathcal{Q}]\!]_\psi$ for all $w \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, \psi)$, which implies that $\mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, \psi) = \bigcup_{w \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, \psi)} \mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, g_w(\phi_w))$. Since $[\![\mathcal{P}]\!]_\phi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\phi$ for any $\phi \in \Phi$, we know that $[\![\mathcal{P}]\!]_{g(\phi)} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{g(\phi)}$ for all sortwise injections g from $\mathrm{Im}(\phi)$, too. Hence, $[\![\mathcal{P}]\!]_{g_w(\phi_w)} \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_{g_w(\phi_w)}$ for all $w \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, \psi)$. By Lemma 26, it implies that $[\![\mathcal{P}]\!]_\psi \preceq_{\mathrm{tr}} [\![\mathcal{Q}]\!]_\psi$, too. Therefore, Φ is a cut-off set. □

By applying Definition 30 and Proposition 31, we can show that the set of six valuations given in Example 20 is a finite basis, and hence a cut-off set, of our generalised Raft model. However, this is not easy to see, since we basically need to apply Definition 30 an infinite number of times. That is why, in the next section, we will develop an algorithm for the purpose of computing a minimal basis, i.e., the most optimal cut-off set our PR technique can provide.

6. Optimal Cut-Off Algorithm

In this section, we convert the problem of determining a cut-off set into the unsatisfiability in FOL, introduce a new SMT-based semi-algorithm for computing such a set, and prove that the computed cut-off sets are optimal with respect to the PR technique. Hence, the new algorithm is called the optimal cut-off algorithm. Results similar to Lemmata 33, 36, 42, Proposition 37, and Theorem 43 are presented in the conference version of this work [1], but here the results are adapted to the new optimal cut-off algorithm. We also provide the full proofs of Lemmata 33, 42 and Theorem 43 which were not included in [1]. The optimal cut-off algorithm and the results, Lemmata 35, 38, 40 and Proposition 39, related to the proof of optimality are completely new.

In order to introduce an SMT-based optimal cut-off algorithm, we first extend the notion of a branch to PLTSs as a formula in the $\exists^*$-fragment of FOL (Definition 32) and prove a correspondence between the branch strings and the branch formulae (Lemma 33). By using this correspondence, we convert the condition for a finite basis from the level of branch strings to the level of branch formulae (Lemma 35). After that, we convert the subvaluation test $g(\phi) \subseteq_{\mathbb{T},\Pi,\Xi} \psi$ occurring in the definition of a basis into a quantifier-free formula (Lemma 36). This makes it possible to express the condition for a finite basis as the unsatisfiability of a first-order formula (Proposition 37).

The minimal basis is obtained as the set of minimal valuations that satisfy the topology formula and a branch formula (Lemma 40). This means that we need to be able to minimise valuations satisfying the formula of Proposition 37 which is done in two phases, first with respect to the size of the sorts and then with respect to the values of predicates. For that purpose, we convert the related proper subvaluation tests into quantifier-free formulae (Lemma 38). After that, we are able to express the questions concerning the existence of a strictly smaller satisfying valuation (with respect to the size of the sorts and with respect to the values of the predicates) as the satisfiability of a first-order formula (Proposition 39). By combining the results above, we are able to create an SMT-based algorithm that iteratively computes minimal satisfying valuations until the condition for an insufficient basis becomes unsatisfiable (Algorithm 1). The correctness of the algorithm is stated in Theorem 43.

In order to formalise the results, we assume that the set of all variables, including the Boolean ones, $\mathbb{X} \cup \mathbb{F}_0$, is partitioned into sets $\mathbb{X}^0$, $\mathbb{X}'$, and $\mathbb{X}''$, each containing infinitely many variables for each sort. Moreover, we assume that only the variables in $\mathbb{X}^0$ are used in PLTSs and topology formulae and for each atom $a \in \mathbb{A}$ there is a unique variable $x''_a \in \mathbb{X}''$ of the sort T_a. We can now extend the notion of a structure to PLTSs. The structure of a PLTS describes the full branches, from the root to a leaf, in the syntax tree of the PLTS, where the branches are represented as formulae in the $\exists^*$-fragment of FOL. The correspondence between the branch strings and the branch formulae is stated in Lemma 33.

Definition 32 (Structure of PLTS/Branch formula). The *structure (of a PLTS $\mathcal{P}$)* is a finite set of first-order formulae denoted $\mathrm{str}(\mathcal{P})$ and defined inductively as follows:

1. $\mathrm{str}(L[x_1/a_1,\ldots,x_n/a_n]) = \{\top\}$,
2. $\mathrm{str}([G]\,\mathcal{P}') = \{G \wedge \mathcal{B} \mid \mathcal{B} \in \mathrm{str}(\mathcal{P}')\}$,
3. $\mathrm{str}(\mathcal{P}_1 \parallel \mathcal{P}_2) = \{\neg b' \wedge \mathcal{B} \mid \mathcal{B} \in \mathrm{str}(\mathcal{P}_1)\} \cup \{b' \wedge \mathcal{B} \mid \mathcal{B} \in \mathrm{str}(\mathcal{P}_2)\}$, where $b' \in \mathbb{X}' \cap \mathbb{F}_0$ is a Boolean variable not occurring in $\mathrm{str}(\mathcal{P}_1) \cup \mathrm{str}(\mathcal{P}_2)$,
4. $\mathrm{str}(\|_x\,\mathcal{P}') = \{\exists x.(x' = x \wedge \mathcal{B}) \mid \mathcal{B} \in \mathrm{str}(\mathcal{P}')\}$, where $x' \in \mathbb{X}'$ is a variable of the sort T_x not occurring in $\mathrm{str}(\mathcal{P}')$, and
5. $\mathrm{str}(\mathcal{P}' \setminus C) = \mathrm{str}(\mathcal{P}')$.

The elements of $\mathrm{str}(\mathcal{P})$ are called *branch formulae.*

Lemma 33. *Let $\mathcal{P}$ be a PLTS and ϕ a compatible valuation.*

1. *If $\mathcal{B} \in \mathrm{str}(\mathcal{P})$ and $\phi' \in \mathrm{ext}(\phi, \mathrm{sig}_{\mathbb{X}'}(\mathcal{B}))$ is a valuation satisfying $\mathcal{B}$, then $\phi'(y'_1)\cdots\phi'(y'_m) \in \mathrm{str}(\mathcal{P},\phi)$, where $y'_1,\ldots,y'_m$ are the primed variables occurring in $\mathcal{B}$ in this order.*
2. *If $w \in \mathrm{str}(\mathcal{P},\phi)$, then there is $\mathcal{B} \in \mathrm{str}(\mathcal{P})$ and a valuation $\phi' \in \mathrm{ext}(\phi, \mathrm{sig}_{\mathbb{X}'}(\mathcal{B}))$ satisfying $\mathcal{B}$ such that $w = \phi'(y'_1)\cdots\phi'(y'_m)$, where $y'_1,\ldots,y'_m$ are the primed variables occurring in $\mathcal{B}$ in this order.*

Proof. We argue by induction on the structure of $\mathcal{P}$ by using the lemma as the induction hypothesis.

1. In the base step, $\mathcal{P}$ is an elementary PLTS. If $\mathcal{B} \in \mathrm{str}(\mathcal{P})$ and $\phi' \in \mathrm{ext}(\phi, \mathrm{sig}_{\mathbb{X}'}(\mathcal{B}))$ is a valuation satisfying $\mathcal{B}$, then $\mathcal{B} = \top$ and $\phi' = \phi$. Since no (primed) variable occurs in $\mathcal{B}$, $\phi'(y'_1)\cdots\phi'(y'_m)$ is the empty string ε which obviously is in $\mathrm{str}(\mathcal{P},\phi)$.

   In the induction step, we have four cases to consider. First, if $\mathcal{P}$ is $([G]\,\mathcal{P}')$, $\mathcal{B} \in \mathrm{str}(\mathcal{P})$, and $\phi' \in \mathrm{ext}(\phi, \mathrm{sig}_{\mathbb{X}'}(\mathcal{B}))$ is a valuation satisfying $\mathcal{B}$, then $\mathcal{B} = G \wedge \mathcal{B}'$ for some $\mathcal{B}' \in \mathrm{str}(\mathcal{P}')$ and $\phi' \in \mathrm{ext}(\phi, \mathrm{sig}_{\mathbb{X}'}(\mathcal{B}'))$ is a valuation satisfying $\mathcal{B}'$. By the induction hypothesis, it implies that $\phi'(y'_1)\cdots\phi'(y'_m) \in \mathrm{str}(\mathcal{P}',\phi)$, where $y'_1,\ldots,y'_m$ are the primed variables occurring in $\mathcal{B}'$ in this order. Since ϕ' satisfies G and no primed variable occurs in G, also ϕ satisfies G, $\mathrm{str}(\mathcal{P},\phi) = \mathrm{str}(\mathcal{P}',\phi)$, $\phi'(y'_1)\cdots\phi'(y'_m) \in \mathrm{str}(\mathcal{P},\phi)$, and $y'_1,\ldots,y'_m$ are the primed variables occurring in $\mathcal{B}$ in this order.

   Second, if $\mathcal{P}$ is $\mathcal{P}_1 \parallel \mathcal{P}_2$, $\mathcal{B} \in \mathrm{str}(\mathcal{P})$, and $\phi' \in \mathrm{ext}(\phi, \mathrm{sig}_{\mathbb{X}'}(\mathcal{B}))$ is a valuation satisfying $\mathcal{B}$, then $\mathcal{B}$ is $\neg b' \wedge \mathcal{B}'$ for some $\mathcal{B}' \in \mathrm{str}(\mathcal{P}_1)$ or $b' \wedge \mathcal{B}'$ for some $\mathcal{B}' \in \mathrm{str}(\mathcal{P}_2)$, where $b' \in \mathbb{X}' \cap \mathbb{F}_0$ is a Boolean variable not occurring in $\mathrm{str}(\mathcal{P}_1) \cup \mathrm{str}(\mathcal{P}_2)$. If $\mathcal{B}$ is $b' \wedge \mathcal{B}'$ and $\mathcal{B}' \in \mathrm{str}(\mathcal{P}_2)$, then $\phi'' := \phi'|_{\mathrm{dom}(\phi')\setminus\{b'\}} \in \mathrm{ext}(\phi, \mathrm{sig}_{\mathbb{X}'}(\mathcal{B}'))$. By the induction hypothesis, it implies that $\phi''(y'_1)\cdots\phi''(y'_m) \in \mathrm{str}(\mathcal{P}_2,\phi)$, where $y'_1,\ldots,y'_m$ are the primed variables occurring in $\mathcal{B}'$ in this order. Now, $b', y'_1,\ldots,y'_m$ are the primed variables occurring in $\mathcal{B}$ in this order and since ϕ' satisfies b', $\phi'(b')\phi'(y'_1)\cdots\phi'(y'_m) = \mathit{true}\,\phi'(y'_1)\cdots\phi'(y'_m)$ which, by Definition 21, is in $\mathrm{str}(\mathcal{P},\phi)$. The case when $\mathcal{B}$ is $\neg b' \wedge \mathcal{B}'$ and $\mathcal{B}' \in \mathrm{str}(\mathcal{P}_1)$ is similar.

   Third, if $\mathcal{P}$ is $\|_x\,\mathcal{P}'$, $\mathcal{B} \in \mathrm{str}(\mathcal{P})$, and $\phi' \in \mathrm{ext}(\phi, \mathrm{sig}_{\mathbb{X}'}(\mathcal{B}))$ is a valuation satisfying $\mathcal{B}$, then $\mathcal{B} = \exists x.(x' = x \wedge \mathcal{B}')$ for some $\mathcal{B}' \in \mathrm{str}(\mathcal{P}')$ and some variable $x' \in \mathbb{X}'$ not occurring in $\mathrm{str}(\mathcal{P}')$. Let ϕ_1 be a valuation with the domain $\mathrm{dom}(\phi) \cup \{x\}$ such that $\phi_1(x) = \phi'(x')$ and the valuations ϕ_1 and ϕ agree elsewhere in the domain. Then $\phi_1 \in \mathrm{ext}(\phi, \{x\})$ is a valuation compatible with $\mathcal{P}'$. Let ϕ'_1 be a valuation with the domain $\mathrm{dom}(\phi_1) \cup \mathrm{sig}_{\mathbb{X}'}(\mathcal{B}')$ such that $\phi'_1(x') = \phi'(x')$ for all $x' \in \mathrm{sig}_{\mathbb{X}'}(\mathcal{B}')$ and the valuations ϕ'_1 and ϕ_1 agree elsewhere in the domain. Then $\phi'_1 \in \mathrm{ext}(\phi_1, \mathrm{sig}_{\mathbb{X}'}(\mathcal{B}'))$. By the induction hypothesis, it implies that $\phi'_1(y'_1)\cdots\phi'_1(y'_m) \in \mathrm{str}(\mathcal{P}',\phi_1)$, where $y'_1,\ldots,y'_m$ are the primed variables occurring in $\mathcal{B}'$ in this order. Now, $x', y'_1,\ldots,y'_m$ are the primed variables occurring in $\mathcal{B}$ in this order and $\phi'(x')\phi'(y'_1)\cdots\phi'(y'_m) = \phi_1(x)\phi'_1(y'_1)\cdots\phi'_1(y'_m)$ which, by Definition 21, is in $\mathrm{str}(\mathcal{P},\phi)$.

   Finally, the case when $\mathcal{P}$ is $\mathcal{P}' \setminus C$ is trivial, because $\mathrm{str}(\mathcal{P}' \setminus C) = \mathrm{str}(\mathcal{P}')$ and $\mathrm{str}(\mathcal{P}' \setminus C,\phi) = \mathrm{str}(\mathcal{P}',\phi)$. Therefore, by the induction principle, the first claim of the lemma holds.
2. In the base step, $\mathcal{P}$ is an elementary PLTS. If $w \in \mathrm{str}(\mathcal{P},\phi)$, then $w = \varepsilon$. Now, $\mathrm{str}(\mathcal{P})$ contains a single element $\top$ and ϕ is a valuation in $\mathrm{ext}(\phi, \mathrm{sig}_{\mathbb{X}'}(\top))$ satisfying $\top$ such that w is the string consisting of the values of the primed variables in $\top$.

In the induction step, we have four cases to consider. First, if $\mathcal{P}$ is $([G]\,\mathcal{P}')$ and $w \in \mathrm{str}(\mathcal{P}, \phi)$, then ϕ must satisfy G and w must be in $\mathrm{str}(\mathcal{P}', \phi)$. By the induction hypothesis, there is $\mathcal{B}' \in \mathrm{str}(\mathcal{P}')$ and a valuation $\phi' \in \mathrm{ext}(\phi, \mathrm{sig}_{\mathbb{X}'}(\mathcal{B}'))$ satisfying $\mathcal{B}'$ such that $w = \phi'(y'_1) \cdots \phi'(y'_m)$, where $y'_1, \ldots, y'_m$ are the primed variables occurring in $\mathcal{B}'$ in this order. It means that $G \wedge \mathcal{B}' \in \mathrm{str}(\mathcal{P})$, ϕ' is a valuation in $\mathrm{ext}(\phi, \mathrm{sig}_{\mathbb{X}'}(G \wedge \mathcal{B}'))$ satisfying $G \wedge \mathcal{B}'$, and $y'_1, \ldots, y'_m$ are the primed variables occurring in $G \wedge \mathcal{B}'$ in this order such that $w = \phi'(y'_1) \cdots \phi'(y'_m)$.

Second, if $\mathcal{P}$ is $\mathcal{P}_1 \parallel \mathcal{P}_2$ and $w \in \mathrm{str}(\mathcal{P}, \phi)$, then either $w = \mathit{false}\, w'$ and $w' \in \mathrm{str}(\mathcal{P}_1, \phi)$ or $w = \mathit{true}\, w'$ and $w' \in \mathrm{str}(\mathcal{P}_2, \phi)$. Let us assume the latter, $w = \mathit{true}\, w'$ and $w' \in \mathrm{str}(\mathcal{P}_2, \phi)$. By the induction hypothesis, there is $\mathcal{B}' \in \mathrm{str}(\mathcal{P}_2)$ and a valuation $\phi'' \in \mathrm{ext}(\phi, \mathrm{sig}_{\mathbb{X}'}(\mathcal{B}'))$ satisfying $\mathcal{B}'$ such that $w' = \phi''(y'_1) \cdots \phi''(y'_m)$, where $y'_1, \ldots, y'_m$ are the primed variables occurring in $\mathcal{B}'$ in this order. Now, $b' \wedge \mathcal{B}' \in \mathrm{str}(\mathcal{P})$ for some Boolean variable $b' \in \mathbb{X}' \cap \mathbb{F}_0$ not occurring in $\mathrm{str}(\mathcal{P}_1) \cup \mathrm{str}(\mathcal{P}_2)$ and $\phi' := \phi'' \cup \{(b', \mathit{true})\}$ is a valuation in $\mathrm{ext}(\phi, \mathrm{sig}_{\mathbb{X}'}(b' \wedge \mathcal{B}'))$ satisfying $b' \wedge \mathcal{B}'$. Moreover, $b', y'_1, \ldots, y'_m$ are the primed variables occurring in $b' \wedge \mathcal{B}'$ in this order such that $w = \phi'(b')\phi'(y'_1) \cdots \phi'(y'_m)$. The case when $w = \mathit{false}\, w'$ and $w' \in \mathrm{str}(\mathcal{P}_1, \phi)$ is similar.

Third, if $\mathcal{P}$ is $\|_x \mathcal{P}'$ and $w \in \mathrm{str}(\mathcal{P}, \phi)$, then $w = \phi_1(x)w'$ for some valuation $\phi_1 \in \mathrm{ext}(\phi, \{x\})$ and $w' \in \mathrm{str}(\mathcal{P}', \phi_1)$. By the induction hypothesis, there is $\mathcal{B}' \in \mathrm{str}(\mathcal{P}')$ and a valuation $\phi'_1 \in \mathrm{ext}(\phi_1, \mathrm{sig}_{\mathbb{X}'}(\mathcal{B}'))$ satisfying $\mathcal{B}'$ such that $w' = \phi'_1(y'_1) \cdots \phi'_1(y'_m)$, where $y'_1, \ldots, y'_m$ are the primed variables occurring in $\mathcal{B}'$ in this order. It means that $\exists x.(x' = x \wedge \mathcal{B}') \in \mathrm{str}(\mathcal{P})$ for some variable $x' \in \mathbb{X}'$ not occurring in $\mathrm{str}(\mathcal{P}')$ and $\phi' := \phi'_1|_{\mathrm{dom}(\phi'_1) \setminus \{x\}} \cup \{(x', \phi_1(x))\}$ is a valuation in $\mathrm{ext}(\phi, \mathrm{sig}_{\mathbb{X}'}(\exists x.(x' = x \wedge \mathcal{B}')))$ satisfying $\exists x.(x' = x \wedge \mathcal{B}')$. Moreover, $x', y'_1, \ldots, y'_m$ are the primed variables occurring in $\exists x.(x' = x \wedge \mathcal{B}')$ in this order such that $w = \phi_1(x)w' = \phi'(x')\phi'(y'_1) \cdots \phi'(y'_m)$.

Finally, the case when $\mathcal{P}$ is $\mathcal{P}' \setminus C$ is trivial, because $\mathrm{str}(\mathcal{P}' \setminus C) = \mathrm{str}(\mathcal{P}')$ and $\mathrm{str}(\mathcal{P}' \setminus C, \phi) = \mathrm{str}(\mathcal{P}', \phi)$. Therefore, by the induction principle, also the second claim of the lemma holds. □

Example 34. Recall the generalised Raft example and the valuation θ from Example 22. Since the PLTS $(\mathit{Raft}' \parallel \mathit{Spec})$ is composed of three elementary PLTSs, its structure consists of three branch formulae

$$\mathcal{B}_1 := \neg b'_1 \wedge \exists x_0.(x'_0 = x_0 \wedge \neg b'_0 \wedge \exists y.(y' = y \wedge \exists x_1.(x'_1 = x_1 \wedge Q_S(x_0, y, x_1) \wedge \top))) ,$$
$$\mathcal{B}_2 := \neg b'_1 \wedge \exists x_0.(x'_0 = x_0 \wedge b'_0 \wedge \exists x_1.(x'_1 = x_1 \wedge \exists x_2.(x'_2 = x_2 \wedge x_1 \neq x_2 \wedge \exists y.(y' = y \wedge \top)))) , \text{ and}$$
$$\mathcal{B}_3 := b'_1 \wedge \exists x_0.(x'_0 = x_0 \wedge \exists x_1.(x'_1 = x_1 \wedge \exists x_2.(x'_2 = x_2 \wedge \exists y.(y' = y \wedge Q_S(x_0, y, x_2) \wedge Q_S(x_1, y, x_2) \wedge \top)))) .$$

By Lemma 33, we know that every branch string $\mathit{false}\, s_i\, \mathit{false}\, t_l s_j \in \mathrm{str}(\mathit{Raft}' \parallel \mathit{Spec}, \theta)$ corresponds to a valuation $\theta' \in \mathrm{ext}(\theta, \{b'_1, x'_0, b'_0, y', x'_1\})$ satisfying $\mathcal{B}_1$ such that $\theta'(b'_1)\theta'(x'_0)\theta'(b'_0)\theta'(y')\theta'(x'_1) = \mathit{false}\, s_i\, \mathit{false}\, t_l s_j$, for every $\mathit{false}\, s_i\, \mathit{true}\, s_j s_k t_l \in \mathrm{str}(\mathit{Raft}' \parallel \mathit{Spec})$ there is a valuation $\theta' \in \mathrm{ext}(\theta, \{b'_1, x'_0, b'_0, y', x'_1, x'_2\})$ satisfying $\mathcal{B}_2$ such that $\theta'(b'_1)\theta'(x'_0)\theta'(b'_0)\theta'(x'_1)\theta'(x'_2)\theta'(y')$ equals $\mathit{false}\, s_i\, \mathit{true}\, s_j s_k t_l$, and each branch string $\mathit{true}\, s_i s_j s_k t_l \in \mathrm{str}(\mathit{Raft}' \parallel \mathit{Spec}, \theta)$ matches a valuation $\theta' \in \mathrm{ext}(\theta, \{b'_1, x'_0, x'_1, x'_2, y'\})$ satisfying $\mathcal{B}_3$ such that $\theta'(b'_1)\theta'(x'_0)\theta'(x'_1)\theta'(x'_2)\theta'(y') = \mathit{true}\, s_i s_j s_k t_l$. □

By exploiting the correspondence between the branch strings and the branch formulae, we can convert the definition of a basis from the level of branch strings to the level of branch formulae.

Lemma 35. *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$ be a parameterised system and $\Phi \subseteq \mathrm{va}(\mathcal{F} \mid \mathcal{P}, \mathcal{Q})$ a set of valuations. For every branch formula $\mathcal{B} \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q})$, let $\Phi_{\mathcal{B}} := \{\phi' \in \mathrm{ext}(\phi, \mathrm{sig}_{\mathbb{X}'}(\mathcal{B})) \mid \phi \in \Phi, [\![\mathcal{B}]\!]_{\phi'}\}$ be the set of the extensions of the valuations in Φ to $\mathrm{sig}_{\mathbb{X}'}(\mathcal{B})$ satisfying $\mathcal{B}$. The set Φ is a basis of $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$ if and only if for every $\mathcal{B} \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q})$ and every valuation $\psi \in \mathrm{va}(\mathcal{F} \wedge \mathcal{B} \mid \mathcal{P}, \mathcal{Q})$ there is a valuation $\phi \in \Phi_{\mathcal{B}}$ and a sortwise injection $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\psi)$ such that $g(\phi)$ is a $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of ψ satisfying $\mathcal{F} \wedge \mathcal{B}$.*

Proof. Let us first assume that Φ is a basis of $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$, $\mathcal{B} \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q})$, and $\psi' \in \mathrm{va}(\mathcal{F} \wedge \mathcal{B} \mid \mathcal{P}, \mathcal{Q})$. Let ψ be the restriction of ψ' to $\mathrm{dom}(\mathcal{P} \parallel \mathcal{Q}) \cup \mathrm{dom}(\mathcal{F})$. By Item 1 of Lemma 33, $\psi'(y'_1) \cdots \psi'(y'_m) \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, \psi)$, where $y'_1, \ldots, y'_m$ are the primed variables occurring in $\mathcal{B}$ in this order. Since Φ is a basis, there is $\phi \in \Phi$ and a sortwise injection g from $\mathrm{Im}(\phi)$ such that $\psi'(y'_1) \cdots \psi'(y'_m) \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, g(\phi))$ and $g(\phi)$ is a $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of ψ. By Item 2 of Lemma 33, there is $\mathcal{B}' \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q})$ and a valuation $\phi' \in \mathrm{ext}(g(\phi), \mathrm{sig}_{\mathbb{X}'}(\mathcal{B}'))$ satisfying $\mathcal{B}'$ such that $\psi'(y'_1) \cdots \psi'(y'_m) = \phi'(z'_1) \cdots \phi'(z'_m)$, where $z'_1, \ldots, z'_m$ are

the primed variables occurring in $\mathcal{B}'$ in this order. By induction on the structure of a PLTS, one can prove that this is only possible if $y_i = z_i$ for all $i \in \{1, \ldots, m\}$ and $\mathcal{B} = \mathcal{B}'$. Since $\phi' \in \mathrm{ext}(g(\phi), \mathrm{sig}_{\mathbb{X}'}(\mathcal{B}'))$ satisfies $\mathcal{B}'$, this implies that $g^{-1}(\phi') \in \mathrm{ext}(\phi, \mathrm{sig}_{\mathbb{X}'}(\mathcal{B}))$ satisfies $\mathcal{B}$. As $\phi \in \Phi$, it means that $g^{-1}(\phi') \in \Phi_{\mathcal{B}}$. Moreover, since $g(\phi)$ is a $(\mathbb{T}, \mathcal{P} \,\|\, \mathcal{Q})$-subvaluation of ψ, $\phi' \in \mathrm{ext}(g(\phi), \mathrm{sig}_{\mathbb{X}'}(\mathcal{B}))$, $\psi' \in \mathrm{ext}(\psi, \mathrm{sig}_{\mathbb{X}'}(\mathcal{B}))$, and the valuations ϕ' and ψ' agree on the values of the primed variables occurring in $\mathcal{B}$, ϕ' is a $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of ψ'. Finally, because $g^{-1}(\phi') \in \Phi_{\mathcal{B}}$, it means that $g^{-1}(\phi')$ and therefore also $g(g^{-1}(\phi'))$ satisfy $\mathcal{F} \wedge \mathcal{B}$. Hence, $g^{-1}(\phi')$ is the valuation in $\Phi_{\mathcal{B}}$ and g the sortwise injection such that $g(g^{-1}(\phi'))$ is a $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of ψ' satisfying $\mathcal{F} \wedge \mathcal{B}$.

Let us then assume that for every $\mathcal{B} \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q})$ and every valuation $\psi \in \mathrm{va}(\mathcal{F} \wedge \mathcal{B} \mid \mathcal{P}, \mathcal{Q})$ there is a valuation $\phi \in \Phi_{\mathcal{B}}$ and a sortwise injection $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\psi)$ such that $g(\phi)$ is a $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of ψ satisfying $\mathcal{F} \wedge \mathcal{B}$. In order to show that Φ is a basis, let $\psi \in \mathrm{va}(\mathcal{F} \mid \mathcal{P}, \mathcal{Q})$ and $w \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, \psi)$. By Item 2 of Lemma 33, there is $\mathcal{B} \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q})$ and a valuation $\psi' \in \mathrm{ext}(\psi, \mathrm{sig}_{\mathbb{X}'}(\mathcal{B}))$ satisfying $\mathcal{B}$ such that $w = \psi'(y'_1) \cdots \psi'(y'_m)$, where $y'_1, \ldots, y'_m$ are the primed variables occurring in $\mathcal{B}$ in this order. By the assumption, there is a valuation $\phi' \in \Phi_{\mathcal{B}}$ and a sortwise injection $g : \mathrm{Im}(\phi') \to \mathrm{Im}(\psi')$ such that $g(\phi')$ is a $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of ψ' satisfying $\mathcal{F} \wedge \mathcal{B}$. Hence, there is $\phi \in \Phi$ such that $g(\phi') \in \mathrm{ext}(g(\phi), \mathrm{sig}_{\mathbb{X}'}(\mathcal{B}))$, which by Item 1 of Lemma 33 implies that $(g(\phi'))(y'_1) \cdots (g(\phi'))(y'_m) \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, g(\phi))$. Since $g(\phi')$ is a $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of ψ', it means that $\psi'(y_i) = (g(\phi'))(y_i)$ for all $i \in \{1, \ldots, m\}$. Therefore, $w = (g(\phi'))(y'_1) \cdots (g(\phi'))(y'_m) \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, g(\phi))$ and also $g(\phi)$ must be a $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of ψ, which proves that Φ is indeed a basis. □

Lemma 35 gives a sufficient and necessary condition for a basis, but it does not clearly say which valuations should be included in it. That is why we will transform the condition of Lemma 35 into a first-order formula, the satisfaction of which can be, in decidable cases, analysed by using existing tools. For that purpose, we introduce Lemma 36 which tells how the subvaluation test $g(\phi) \subseteq_{\mathbb{T},\Pi,\Xi} \psi$ occurring in the condition is converted into a quantifier-free formula.

Lemma 36. *Let ϕ and ψ be valuations with the same domain, $\Pi, \Xi \subseteq \mathrm{dom}_{\mathbb{F}}(\phi)$ sets of predicates, $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\psi)$ a sortwise function, and ψ_g the valuation in $\mathrm{ext}(\psi, \{x''_a \mid a \in \mathrm{Im}(\phi)\})$ such that $\psi_g(x''_a) = g(a)$ for all $a \in \mathrm{Im}(\phi)$. Then g is an injection and $g(\phi) \subseteq_{\mathbb{T},\Pi,\Xi} \psi$, if and only if $[\![NoSval(\phi, \Pi, \Xi)]\!]_{\psi_g}$ is false, where*

$$NoSval(\phi, \Pi, \Xi) := \Big(\bigvee_{\substack{\{a,b\} \subseteq \mathrm{Im}(\phi) \\ a \neq b}} x''_a = x''_b \Big) \vee \Big(\bigvee_{x \in \mathrm{dom}_{\mathbb{X}}(\phi)} x''_{\phi(x)} \neq x \Big) \vee$$
$$\Big(\bigvee_{F \in \Pi} \bigvee_{(a_1, \ldots, a_n) \in \phi(F)} \neg F(x''_{a_1}, \ldots, x''_{a_n}) \Big) \vee \Big(\bigvee_{F \in \Xi} \bigvee_{(a_1, \ldots, a_n) \in \overline{\phi(F)}} F(x''_{a_1}, \ldots, x''_{a_n}) \Big) .$$

Proof. Let us first assume that g is an injection and $g(\phi) \subseteq_{\mathbb{T},\Pi,\Xi} \psi$. Because g is an injection, the variables x''_a, where $a \in \mathrm{Im}(\phi)$, representing the image of g, are mapped to different values. Hence, the first big disjunction is *false*. Since $g(\phi) \subseteq_{\mathbb{T},\Pi,\Xi} \psi$, it implies that $\psi_g(x''_{\phi(x)}) = g(\phi(x)) = \psi(x) = \psi_g(x)$ for all $x \in \mathrm{dom}_{\mathbb{X}}(\phi)$. Hence, the second big disjunction is *false*. If $F \in \Pi$ and $(a_1, \ldots, a_n) \in \phi(F)$, then $(g(a_1), \ldots, g(a_n)) \in (g(\phi))(F)$. Since $g(\phi) \subseteq_{\mathbb{T},\Pi,\Xi} \psi$, it implies that $(g(a_1), \ldots, g(a_n)) \in \psi(F)$, too. This is equivalent to $(\psi_g(x''_{a_1}), \ldots, \psi_g(x''_{a_n})) \in \psi_g(F)$, which means that also the third big disjunction is *false*. Similarly, we can show that the fourth big disjunction and hence $[\![NoSval(\phi, \Pi, \Xi)]\!]_{\psi_g}$ are *false*, too.

Let us then assume that $[\![NoSval(\phi, \Pi, \Xi)]\!]_{\psi_g}$ is *false*. Because the first big disjunction is *false*, it implies that the variables x''_a, where $a \in \mathrm{Im}(\phi)$, representing the image of g are mapped to different values. Hence, g is an injection. In order to prove that $g(\phi) \subseteq_{\mathbb{T},\Pi,\Xi} \psi$, we will show that the conditions (1)–(6) of Definition 23 are met. (1) By the assumption, $\mathrm{dom}(g(\phi)) = \mathrm{dom}(\phi) = \mathrm{dom}(\psi)$. (2) Because g is a sortwise function: $\mathrm{Im}(\phi) \to \mathrm{Im}(\psi)$, it implies that $(g(\phi))(T) \subseteq \psi(T)$ for all sorts $T \in \mathrm{dom}_{\mathbb{T}}(\phi)$. (3) Trivial, because $\mathrm{dom}_{\mathbb{T}}(\phi) \setminus \mathbb{T}$ is empty. (4) Since the second big disjunction is *false*, it means that $(g(\phi))(x) = g(\phi(x)) = \psi_g(x''_{\phi(x)}) = \psi_g(x) = \psi(x)$ for all $x \in \mathrm{dom}_{\mathbb{X}}(\phi)$, (5) If $F \in \Pi$ and $(a_1, \ldots, a_n) \in (g(\phi))(F)$, then $(g^{-1}(a_1), \ldots, g^{-1}(a_n)) \in \phi(F)$. Because the third big disjunction is *false*, it implies that $(\psi_g(x''_{g^{-1}(a_1)}), \ldots, \psi_g(x''_{g^{-1}(a_n)})) \in \psi_g(F)$.

Since $(\psi_g(x''_{g^{-1}(a_1)}), \ldots, \psi_g(x''_{g^{-1}(a_n)})) = (g(g^{-1}(a_1)), \ldots, g(g^{-1}(a_n))) = (a_1, \ldots, a_n)$ and $\psi_g(F) = \psi(F)$, it means that $(a_1, \ldots, a_n) \in \psi(F)$. (6) Similar to (5). □

By combining Lemmata 33 and 36, we can now convert the sufficient and necessary condition for a finite basis into the unsatisfiability of a first-order formula.

Proposition 37 (Finite basis proposition). *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$ be a parameterised system and Φ a finite set of valuations in $\mathrm{va}(\mathcal{F} \mid \mathcal{P}, \mathcal{Q})$.*

1. *The set Φ is a finite basis of $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$, if and only if the first-order formula*

$$\mathit{NoBas}(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi) := \mathcal{F} \wedge \mathcal{B} \wedge \bigwedge_{\phi \in \Phi_{\mathcal{B}}} \left(\forall x''_{a_1}. \cdots . \forall x''_{a_n}. \mathit{NoSval}(\phi, \mathrm{pr}^+(\mathcal{P} \parallel \mathcal{Q}), \mathrm{pr}^-(\mathcal{P} \parallel \mathcal{Q}))\right) \quad (1)$$

   is unsatisfiable for all branch formulae $\mathcal{B} \in \mathrm{str}(\mathcal{P} \| \mathcal{Q})$, where $\Phi_{\mathcal{B}} = \{\phi' \in \mathrm{ext}(\phi, \mathrm{sig}_{\mathbb{X}'}(\mathcal{B})) \mid \phi \in \Phi, [\![\mathcal{B}]\!]_{\phi'}\}$ as in Lemma 35 and for every $\phi \in \Phi_{\mathcal{B}}$, $a_1, \ldots, a_n$ are the atoms in $\mathrm{Im}(\phi)$.

2. *If $\psi \in \mathrm{va}(\mathcal{F} \wedge \mathcal{B} \mid \mathcal{P}, \mathcal{Q})$ is a minimal valuation w.r.t. the $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation order (modulo isomorphism) satisfying $\mathit{NoBas}(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi)$ for some branch formula $\mathcal{B} \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q})$, then ψ is a minimal valuation w.r.t. the $(\mathbb{T}, \mathcal{P} \| \mathcal{Q})$-subvaluation order (modulo isomorphism) satisfying $\mathcal{F} \wedge \mathcal{B}$ such that for every valuation $\phi \in \Phi_{\mathcal{B}}$ and for every sortwise injection $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\psi)$, $g(\phi)$ is not a $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of ψ.*

Proof.

1. By Lemma 35, Φ is not a basis if and only if the following condition holds: There is a branch formula $\mathcal{B} \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q})$ and a valuation $\psi \in \mathrm{va}(\mathcal{F} \wedge \mathcal{B} \mid \mathcal{P}, \mathcal{Q})$ such that for every valuation $\phi \in \Phi_{\mathcal{B}}$ and for every sortwise function $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\psi)$, g is not an injection, $g(\phi)$ is not a $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of ψ, or $g(\phi)$ does not satisfy $\mathcal{F} \wedge \mathcal{B}$. Note that $\phi \in \Phi_{\mathcal{B}}$ implies that ϕ satisfies $\mathcal{F} \wedge \mathcal{B}$. Hence, if g is an injection, then by Lemma 29, also $g(\phi)$ satisfies $\mathcal{F} \wedge \mathcal{B}$. This means that the last part, $g(\phi)$ does not satisfy $\mathcal{F} \wedge \mathcal{B}$, of the condition can be dropped.

   By applying Lemma 36, we can convert the condition into the following equivalent form: There is a branch formula $\mathcal{B} \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q})$ and a valuation $\psi \in \mathrm{va}(\mathcal{F} \wedge \mathcal{B} \mid \mathcal{P}, \mathcal{Q})$ such that for all valuations $\phi \in \Phi_{\mathcal{B}}$ and for all sortwise functions $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\psi)$, $[\![\mathit{NoSval}(\phi, \mathrm{pr}^+(\mathcal{P} \parallel \mathcal{Q}), \mathrm{pr}^-(\mathcal{P} \parallel \mathcal{Q}))]\!]_{\psi_g}$ is *true*.

   Since Φ is finite and each valuation only has finitely many extensions to $\mathrm{sig}_{\mathbb{X}'}(\mathcal{B})$, universal quantification over the valuations in $\Phi_{\mathcal{B}}$ can be substituted by a finite conjunction. Universal quantification over sortwise functions is, in general, a second-order construct. However, since the image of each sortwise function $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\psi)$ is finite and represented by the variables in $\{x''_a \mid a \in \mathrm{Im}(\phi)\}$, the universal quantification over sortwise functions can be replaced by the universal quantification over these variables. Hence, the condition gets the following equivalent form: There is a branch formula $\mathcal{B} \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q})$ and a valuation $\psi \in \mathrm{va}(\mathcal{F} \wedge \mathcal{B} \mid \mathcal{P}, \mathcal{Q})$ such that $[\![\bigwedge_{\phi \in \Phi_{\mathcal{B}}} \left(\forall x''_{a_1}. \cdots . \forall x''_{a_n}. \mathit{NoSval}(\phi, \mathrm{pr}^+(\mathcal{P} \parallel \mathcal{Q}), \mathrm{pr}^-(\mathcal{P} \parallel \mathcal{Q}))\right)]\!]_{\psi}$ is *true*.

   Finally, the existence of $\psi \in \mathrm{va}(\mathcal{F} \wedge \mathcal{B} \mid \mathcal{P}, \mathcal{Q})$ can be simply encoded as the satisfaction of the formula $\mathcal{F} \wedge \mathcal{B}$. This means that Φ is not a basis if and only if there is a branch formula $\mathcal{B} \in \mathrm{str}(\mathcal{P} \| \mathcal{Q})$ such that Formula 1 is satisfiable. Obviously, the formula is also in FOL since it only involves first-order constructs.

2. Let us assume that $\psi \in \mathrm{va}(\mathcal{F} \wedge \mathcal{B} \mid \mathcal{P}, \mathcal{Q})$ is a minimal valuation w.r.t. the $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation order (modulo isomorphism) satisfying $\mathit{NoBas}(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi)$ for some branch formula $\mathcal{B} \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q})$. Obviously, then ψ satisfies $\mathcal{F} \wedge \mathcal{B}$ and $[\![\mathit{NoSval}(\phi, \mathrm{pr}^+(\mathcal{P} \parallel \mathcal{Q}), \mathrm{pr}^-(\mathcal{P} \parallel \mathcal{Q}))]\!]_{\psi'}$ is *true* for every $\phi \in \Phi_{\mathcal{B}}$ and every $\psi' \in \mathrm{ext}(\psi, \{x''_a \mid a \in \mathrm{Im}(\phi)\})$. In other words, $[\![\mathit{NoSval}(\phi, \mathrm{pr}^+(\mathcal{P} \parallel \mathcal{Q}), \mathrm{pr}^-(\mathcal{P} \parallel \mathcal{Q}))]\!]_{\psi_g}$ is *true* for every $\phi \in \Phi_{\mathcal{B}}$ and every sortwise function $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\psi)$. By Lemma 36, it implies that for every $\phi \in \Phi_{\mathcal{B}}$ and every sortwise function $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\psi)$, g is not an injection or $g(\phi)$ is

not a $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of ψ. Hence, for every valuation $\phi \in \Phi$ and for every sortwise injection $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\psi)$, $g(\phi)$ is not a $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of ψ.

We still need to show that ψ is a minimal valuation satisfying $\mathcal{F} \wedge \mathcal{B}$ with this property. For that purpose, let us assume that ψ' is a $(\mathbb{T}, \mathcal{P} \| \mathcal{Q})$-subvaluation of ψ, ψ' satisfies $\mathcal{F} \wedge \mathcal{B}$, and for every valuation $\phi \in \Phi$ and for every sortwise injection $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\psi')$, $g(\phi)$ is not a $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of ψ'. Then for every $\phi \in \Phi_{\mathcal{B}}$ and every sortwise function $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\psi')$, g is not an injection or $g(\phi)$ is not a $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of ψ'. By Lemma 36, it implies that $[\![\mathit{NoSval}(\phi, \mathrm{pr}^+(\mathcal{P} \parallel \mathcal{Q}), \mathrm{pr}^-(\mathcal{P} \parallel \mathcal{Q}))]\!]_{\psi_g}$ is *true* for every $\phi \in \Phi_{\mathcal{B}}$ and every sortwise function $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\psi')$. In other words, $[\![\mathit{NoSval}(\phi, \mathrm{pr}^+(\mathcal{P} \parallel \mathcal{Q}), \mathrm{pr}^-(\mathcal{P} \parallel \mathcal{Q}))]\!]_{\psi''}$ is *true* for every $\phi \in \Phi_{\mathcal{B}}$ and every $\psi'' \in \mathrm{ext}(\psi', \{x''_a \mid a \in \mathrm{Im}(\phi)\})$, which means that $[\![\bigwedge_{\phi \in \Phi_{\mathcal{B}}} (\forall x''_{a_1} . \cdots . \forall x''_{a_n} . \mathit{NoSval}(\phi, \mathrm{pr}^+(\mathcal{P} \parallel \mathcal{Q}), \mathrm{pr}^-(\mathcal{P} \parallel \mathcal{Q})))]\!]_{\psi'}$ is *true*. Therefore, ψ' is a valuation satisfying $\mathit{NoBas}(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi)$ and smaller than or equal to ψ. Since ψ is a minimal valuation satisfying $\mathit{NoBas}(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi)$, it implies that $\psi' = \psi$. Hence, ψ is a minimal valuation w.r.t. the $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation order (modulo isomorphism) satisfying $\mathcal{F} \wedge \mathcal{B}$ such that for every valuation $\phi \in \Phi$ and for every sortwise injection $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\psi)$, $g(\phi)$ is not a $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of ψ. □

The iterative application of Proposition 37 allows us to determine a finite basis of a parameterised system. However, in order to compute a minimal basis, we need to be able to minimise valuations satisfying Formula 1. This is done in two phases, first with respect to the size of the sorts and then with respect to the values of the predicates. Since we want to automate the minimisation phase, too, we convert the question on the existence of a strictly smaller satisfying valuation into a first-order formula, the satisfaction of which can be analysed by using an SMT solver. For that purpose, we need to convert the related proper subvaluation tests into quantifier-free formulae.

Lemma 38. *Let ϕ and ϕ' be valuations with the same domain, $\Pi, \Xi \subseteq \mathrm{dom}_{\mathbb{F}}(\phi)$ sets of predicates, $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\phi')$ a sortwise function, $\phi'_g \in \mathrm{ext}(\phi', \{x''_a \mid a \in \mathrm{Im}(\phi)\})$ a valuation such that $\phi'_g(x''_a) = g(a)$ for all $a \in \mathrm{Im}(\phi)$, and x'_T a variable of the sort T for each $T \in \mathrm{dom}_{\mathbb{T}}(\phi)$.*

1. *The function g is not an injection and ϕ' is a $(\emptyset, \emptyset, \emptyset)$-subvaluation of $g(\phi)$, i.e., ϕ' and $g(\phi)$ agree on the values of the sorts and variables while the values of the predicates are omitted, if and only if $[\![\mathit{Sval}(\phi)]\!]_{\phi'_g}$ is true, where*

$$\mathit{Sval}(\phi) := \Big(\bigvee_{\substack{a,b \in \mathrm{Im}(\phi) \\ a \neq b}} x''_a = x''_b \Big) \wedge \Big(\bigwedge_{T \in \mathrm{dom}_{\mathbb{T}}(\phi)} \big(\forall x'_T . \big(\bigvee_{a \in \mathrm{Im}(\phi)} x''_a = x'_T \big) \big) \Big) \wedge \Big(\bigwedge_{x \in \mathrm{dom}_{\mathbb{X}}(\phi)} x''_{\phi(x)} = x \Big) .$$

2. *The function g is an injection and ϕ' is a proper $(\emptyset, \Pi, \Xi)$-subvaluation of $g(\phi)$, i.e., ϕ' and $g(\phi)$ agree on the values of the sorts and variables, the values of the predicates are contained, and one of the containments is proper, if and only if $[\![\mathit{Sval}(\phi, \Pi, \Xi)]\!]_{\phi'_g}$ is true, where*

$$\begin{aligned} \mathit{Sval}(\phi, \Pi, \Xi) := {} & \Big(\bigwedge_{\substack{a,b \in \mathrm{Im}(\phi) \\ a \neq b}} x''_a \neq x''_b \Big) \wedge \Big(\bigwedge_{T \in \mathrm{dom}_{\mathbb{T}}(\phi)} \big(\forall x'_T . \big(\bigvee_{a \in \mathrm{Im}(\phi)} x''_a = x'_T \big) \big) \Big) \wedge \Big(\bigwedge_{x \in \mathrm{dom}_{\mathbb{X}}(\phi)} x''_{\phi(x)} = x \Big) \wedge \\ & \Big(\bigwedge_{F \in \Pi} \bigwedge_{(a_1, \ldots, a_n) \in \overline{\phi(F)}} \neg F(x''_{a_1}, \ldots, x''_{a_n}) \Big) \wedge \Big(\bigwedge_{F \in \Xi} \bigwedge_{(a_1, \ldots, a_n) \in \phi(F)} F(x''_{a_1}, \ldots, x''_{a_n}) \Big) \wedge \\ & \Big(\Big(\bigvee_{F \in \Pi} \bigvee_{(a_1, \ldots, a_n) \in \phi(F)} \neg F(x''_{a_1}, \ldots, x''_{a_n}) \Big) \vee \Big(\bigvee_{F \in \Xi} \bigvee_{(a_1, \ldots, a_n) \in \overline{\phi(F)}} F(x''_{a_1}, \ldots, x''_{a_n}) \Big) \Big) . \end{aligned}$$

Proof.

1. Let us first assume that g is not an injection and ϕ' is a $(\emptyset, \emptyset, \emptyset)$-subvaluation of $g(\phi)$. Since g is not an injection, there are two distinct variables x''_a and x''_b representing the image of g, where

$a, b \in \mathrm{Im}(\phi)$, that are mapped to the same value. Hence, the first conjunct is *true*. Since $\phi'|_{\mathbb{T}} = g(\phi)|_{\mathbb{T}}$, $\mathrm{Im}(\phi'_g) = \mathrm{Im}(\phi') = \mathrm{Im}(g(\phi)) = \{\phi'_g(x''_a) \mid a \in \mathrm{Im}(\phi)\}$, which means that the values of the variables x''_a, where $a \in \mathrm{Im}(\phi)$, cover the atoms in the image of ϕ'_g. Therefore, the second conjunct is *true*, too. Because $\phi'|_{\mathbb{X}} = g(\phi)|_{\mathbb{X}}$, then $\phi'_g(x''_{\phi(x)}) = g(\phi(x)) = \phi'(x) = \phi'_g(x)$ for all $x \in \mathrm{dom}_{\mathbb{X}}(\phi)$, which makes the third conjunct hold, too. Hence, $[\![Sval(\phi)]\!]_{\phi'_g}$ is *true*.

Let us then assume that $[\![Sval(\phi)]\!]_{\phi'_g}$ is *true*. Since the first conjunct is *true*, there are two different variables x''_a and x''_b representing the image of g, where a, b are different atoms in $\mathrm{Im}(\phi)$, that are mapped to the same value. This implies that g is not an injection. Since the second conjunct holds, the values of the variables x''_a, where $a \in \mathrm{Im}(\phi)$, representing the image of g cover the atoms in the image of ϕ'_g. As g is a sortwise function, this implies that ϕ' and $g(\phi)$ agree on the values of the sorts. Finally, since the last conjunct is *true* as well, $g(\phi(x)) = \phi'_g(x''_{\phi(x)}) = \phi'_g(x) = \phi'(x)$ for all $x \in \mathrm{dom}_{\mathbb{X}}(\phi)$. Hence, ϕ' is a $(\emptyset, \emptyset, \emptyset)$-subvaluation of $g(\phi)$ and the first part of the lemma holds.

2. Let us first assume that g is an injection and ϕ' is a proper $(\emptyset, \Pi, \Xi)$-subvaluation of $g(\phi)$. Since g is an injection, the variables x''_a representing the image of g, where $a \in \mathrm{Im}(\phi)$, are mapped to different values. Hence, the first big conjunction is *true*. Since $\phi'|_{\mathbb{T}} = g(\phi)|_{\mathbb{T}}$, $|\mathrm{Im}(\phi'_g)| = |\mathrm{Im}(\phi')| = |\mathrm{Im}(g(\phi))| = \{\phi'_g(x''_a) \mid a \in \mathrm{Im}(\phi)\}$, which means that the values of the variables x''_a, where $a \in \mathrm{Im}(\phi)$, cover the atoms in the image of ϕ'_g. Therefore, the second big conjunction is *true*, too. Because ϕ' is a $(\emptyset, \Pi, \Xi)$-subvaluation of $g(\phi)$, then $\phi'_g(x''_{\phi(x)}) = g(\phi(x)) = \phi'(x) = \phi'_g(x)$ for all $x \in \mathrm{dom}_{\mathbb{X}}(\phi)$, which makes the third big conjunction hold, too. Let us then assume that $F \in \Pi$ and $(a_1, \ldots, a_n) \in \overline{\phi(F)}$, which implies that $(g(a_1), \ldots, g(a_n)) \notin (g(\phi))(F)$. Since ϕ' is a $(\emptyset, \Pi, \Xi)$-subvaluation of $g(\phi)$, it implies that $(g(a_1), \ldots, g(a_n)) \notin \phi'(F)$. Hence, $(\phi'_g(x''_{a_1}), \ldots, \phi'_g(x''_{a_n})) = (g(a_1), \ldots, g(a_n)) \notin \phi'(F) = \phi'_g(F)$. Therefore, the fourth big conjunction is *true* as well. Similarly, we can prove that the fifth big conjunction holds, too. Finally, since ϕ' is a proper $(\emptyset, \Pi, \Xi)$-subvaluation of $g(\phi)$, there is $F \in \Pi$ and $(a_1, \ldots, a_n) \in \phi(F)$ such that $(g(a_1), \ldots, g(a_n)) \notin \phi'(F)$ or there is $F \in \Xi$ and $(a_1, \ldots, a_n) \in \overline{\phi(F)}$ such that $(g(a_1), \ldots, g(a_n)) \in \phi'(F)$. If the former holds, then $(\phi'_g(x''_{a_1}), \ldots, \phi'_g(x''_{a_n})) = (g(a_1), \ldots, g(a_n)) \notin \phi'(F) = \phi'_g(F)$, which implies that the first big disjunction in the last line is *true*. If the latter holds, then $(\phi'_g(x''_{a_1}), \ldots, \phi'_g(x''_{a_n})) = (g(a_1), \ldots, g(a_n)) \in \phi'(F) = \phi'_g(F)$, which implies that the second big disjunction in the last line holds. Hence, $[\![Sval(\phi, \Pi, \Xi)]\!]_{\phi'_g}$ is *true*.

   Let us then assume that $[\![Sval(\phi, \Pi, \Xi)]\!]_{\phi'_g}$ is *true*. Since the first big conjunction is *true*, the variables x''_a representing the image of g, where $a \in \mathrm{Im}(\phi)$, are mapped to different values. Since the second big conjunction holds as well, the values of the variables x''_a, where $a \in \mathrm{Im}(\phi)$, representing the image of g cover the atoms in the image of ϕ'_g. Therefore, g is a sortwise injection and the valuations ϕ' and $g(\phi)$ agree on the values of the sorts, i.e., $\phi'|_{\mathbb{T}} = g(\phi)|_{\mathbb{T}}$. Because the third big conjunct is *true*, $g(\phi(x)) = \phi'_g(x''_{\phi(x)}) = \phi'_g(x) = \phi'(x)$ for all $x \in \mathrm{dom}_{\mathbb{X}}(\phi)$, which means that ϕ' and $g(\phi)$ agree on the values of the variables. Let us then assume that $F \in \Pi$ and $(a_1, \ldots, a_n) \in \overline{(g(\phi))(F)}$, which means that $(g^{-1}(a_1), \ldots, g^{-1}(a_n)) \in \overline{\phi(F)}$. Since the fourth big conjunction holds, it implies that $(\phi'_g(x''_{g^{-1}(a_1)}), \ldots, \phi'_g(x''_{g^{-1}(a_n)})) \notin \phi'_g(F)$. Therefore, $(a_1, \ldots, a_n) = (g(g^{-1}(a_1)), \ldots, g(g^{-1}(a_n))) = (\phi'_g(x''_{g^{-1}(a_1)}), \ldots, \phi'_g(x''_{g^{-1}(a_n)})) \notin \phi'_g(F) = \phi'(F)$, which implies that $\phi'(F) \subseteq (g(\phi))(F)$ for all $F \in \Pi$. Similarly, we can prove that $\overline{\phi'(F)} \subseteq \overline{(g(\phi))(F)}$ for all $F \in \Xi$. Finally, since the formula of the last line holds, it means that there is $F \in \Pi$ and $(a_1, \ldots, a_n) \in \phi(F)$ such that $(\phi'_g(x''_{a_1}), \ldots, \phi'_g(x''_{a_n})) \in \overline{\phi'_g(F)}$ or there is $F \in \Xi$ and $(a_1, \ldots, a_n) \in \overline{\phi(F)}$ such that $(\phi'_g(x''_{a_1}), \ldots, \phi'_g(x''_{a_n})) \in \phi'_g(F)$. In the former case, $(g(a_1), \ldots, g(a_n)) \in (g(\phi))(F)$ but $(g(a_1), \ldots, g(a_n)) = (\phi'_g(x''_{a_1}), \ldots, \phi'_g(x''_{a_n})) \notin \phi'_g(F) = \phi'(F)$, which implies that the containment $\phi'(F) \subset (g(\phi))(F)$ is proper. In the latter case, $(g(a_1), \ldots, g(a_n)) \in \overline{(g(\phi))(F)}$ but $(g(a_1), \ldots, g(a_n)) = (\phi'_g(x''_{a_1}), \ldots, \phi'_g(x''_{a_n})) \notin \overline{\phi'_g(F)} = \overline{\phi'(F)}$, which implies that the containment $\overline{\phi'(F)} \subset \overline{(g(\phi))(F)}$ is proper. Hence, ϕ' is a proper $(\emptyset, \Pi, \Xi)$-subvaluation of $g(\phi)$ and also the second part of the lemma holds. □

By using Lemma 38, the question on the existence of a strictly smaller valuation satisfying Formula 1 of Proposition 37 (with respect to the size of the sorts or with respect to the values of the predicates) can be encoded as the satisfaction of a first-order formula given in Proposition 39.

Proposition 39 (Valuation minimisation proposition). *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$ be a parameterised system and $\Phi \subseteq \mathrm{va}(\mathcal{F} \mid \mathcal{P}, \mathcal{Q})$ a finite set of valuations.*

1. *If $\phi \in \mathrm{va}(\mathcal{F} \wedge \mathcal{B} \mid \mathcal{P}, \mathcal{Q})$ is a valuation satisfying $NoBas(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi)$ for some $\mathcal{B} \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q})$, then ϕ is a minimal valuation with respect to the $(\mathbb{T}, \emptyset, \emptyset)$-subvaluation order (modulo isomorphism) satisfying $NoBas(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi)$, if and only if the first-order formula*

$$SMin(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi, \phi) := NoBas(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi) \wedge \exists x''_{a_1}. \cdots .\exists x''_{a_n}.Sval(\phi) \tag{2}$$

is unsatisfiable, where $a_1, \ldots, a_n$ are the atoms in $\mathrm{Im}(\phi)$.

Moreover, if $\phi' \in \mathrm{va}(\mathcal{F} \wedge \mathcal{B} \mid \mathcal{P}, \mathcal{Q})$ is a valuation satisfying $SMin(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi, \phi)$ for some $\mathcal{B} \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q})$, then ϕ' is strictly smaller than ϕ in the $(\mathbb{T}, \emptyset, \emptyset)$-subvaluation order modulo isomorphism.

2. *If $\phi \in \mathrm{va}(\mathcal{F} \wedge \mathcal{B} \mid \mathcal{P}, \mathcal{Q})$ is a valuation satisfying $NoBas(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi)$ for some $\mathcal{B} \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q})$, then ϕ is a minimal valuation with respect to the $(\emptyset, \mathcal{P} \parallel \mathcal{Q})$-subvaluation order (modulo isomorphism) satisfying $NoBas(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi)$, if and only if the first-order formula*

$$PMin(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi, \phi) := NoBas(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi) \wedge \exists x''_{a_1}. \cdots .\exists x''_{a_n}.Sval(\phi, \mathrm{pr}^+(\mathcal{P} \parallel \mathcal{Q}), \mathrm{pr}^-(\mathcal{P} \parallel \mathcal{Q})) \tag{3}$$

is unsatisfiable, where $a_1, \ldots, a_n$ are the atoms in $\mathrm{Im}(\phi)$.

Moreover, if $\phi' \in \mathrm{va}(\mathcal{F} \wedge \mathcal{B} \mid \mathcal{P}, \mathcal{Q})$ is a valuation satisfying $PMin(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi, \phi)$ for some $\mathcal{B} \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q})$, then ϕ' is strictly smaller than ϕ in the $(\emptyset, \mathcal{P} \parallel \mathcal{Q})$-subvaluation order modulo isomorphism.

Proof.

1. Let us first assume that $\phi \in \mathrm{va}(\mathcal{F} \wedge \mathcal{B} \mid \mathcal{P}, \mathcal{Q})$ is a minimal valuation with respect to the $(\mathbb{T}, \emptyset, \emptyset)$-subvaluation order (modulo isomorphism) satisfying $NoBas(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi)$. This means that there is no satisfying valuation ϕ' of $NoBas(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi)$ and a sortwise non-injective function $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\phi')$ such that ϕ' is a $(\emptyset, \emptyset, \emptyset)$-subvaluation of $g(\phi)$. In other words, if ϕ' is a satisfying valuation of $NoBas(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi)$ and g is a sortwise function: $\mathrm{Im}(\phi) \to \mathrm{Im}(\phi')$, then g is an injection or ϕ' is not a $(\emptyset, \emptyset, \emptyset)$-subvaluation of $g(\phi)$. By Item 1 of Lemma 38, it implies that $[\![Sval(\phi)]\!]_{\phi'_g}$ is *false*. In other words, $[\![\exists x''_{a_1}. \cdots .\exists x''_{a_n}.Sval(\phi)]\!]_{\phi'}$ is *false* for every satisfying valuation ϕ' of $NoBas(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi)$. Therefore, Formula 2 is unsatisfiable.

   Let us then assume that Formula 2 is unsatisfiable. It means that if ϕ' is a satisfying valuation of $NoBas(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi)$, then $[\![\exists x''_{a_1}. \cdots .\exists x''_{a_n}.Sval(\phi)]\!]_{\phi'}$ is *false*. In other words, $[\![Sval(\phi)]\!]_{\phi''}$ is *false* for every $\phi'' \in \mathrm{ext}(\phi', \{x''_a \mid a \in \mathrm{Im}(\phi)\})$. This, in turn, means that if g is a sortwise function: $\mathrm{Im}(\phi) \to \mathrm{Im}(\phi')$, then $[\![Sval(\phi)]\!]_{\phi'_g}$ is *false*. By Item 1 of Lemma 38, it implies that g is an injection or ϕ' is not a $(\emptyset, \emptyset, \emptyset)$-subvaluation of $g(\phi)$. Hence, ϕ' is not a proper $(\mathbb{T}, \emptyset, \emptyset)$-subvaluation of $g(\phi)$ for any sortwise function $g : \mathrm{Im}(\phi) \to \mathbb{A}$. In other words, ϕ is a minimal valuation with respect to the $(\mathbb{T}, \emptyset, \emptyset)$-subvaluation order (modulo isomorphism) satisfying $NoBas(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi)$.

   Finally, we assume that ϕ' is a satisfying valuation of Formula 2. It means that there is a valuation $\phi'' \in \mathrm{ext}(\phi', \{x''_a \mid a \in \mathrm{Im}(\phi)\})$ such that $[\![Sval(\phi)]\!]_{\phi''}$ is *true*. In other words, there is a sortwise function $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\phi')$ such that $[\![Sval(\phi)]\!]_{\phi'_g}$ is *true*. By Item 1 of Lemma 38, it implies that g is not an injection and ϕ' is a $(\emptyset, \emptyset, \emptyset)$-subvaluation of $g(\phi)$. Hence, ϕ' is strictly smaller than ϕ with respect to the $(\mathbb{T}, \emptyset, \emptyset)$-subvaluation order modulo isomorphism.

2. Let us first assume that $\phi \in \mathrm{va}(\mathcal{F} \wedge \mathcal{B} \mid \mathcal{P}, \mathcal{Q})$ is a minimal valuation with respect to the $(\emptyset, \mathcal{P} \parallel \mathcal{Q})$-subvaluation order (modulo isomorphism) satisfying $NoBas(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi)$. This means that if ϕ' is a satisfying valuation of $NoBas(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi)$ and $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\phi')$ a sortwise injection, then ϕ' is not a proper $(\emptyset, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of $g(\phi)$. By Item 2 of Lemma 38, it implies that $[\![Sval(\phi, \mathrm{pr}^+(\mathcal{P} \parallel \mathcal{Q}), \mathrm{pr}^-(\mathcal{P} \parallel \mathcal{Q}))]\!]_{\phi'_g}$ is *false*. Since $[\![Sval(\phi, \mathrm{pr}^+(\mathcal{P} \parallel \mathcal{Q}), \mathrm{pr}^-(\mathcal{P} \parallel \mathcal{Q}))]\!]_{\phi'_g}$ is *false* also when g is not an injection, $[\![\exists x''_{a_1}. \cdots .\exists x''_{a_n}.Sval(\phi, \mathrm{pr}^+(\mathcal{P} \parallel \mathcal{Q}), \mathrm{pr}^-(\mathcal{P} \parallel \mathcal{Q}))]\!]_{\phi'}$ is *false* for every satisfying valuation ϕ' of $NoBas(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi)$. Therefore, Formula 3 is unsatisfiable.

Let us then assume that Formula 3 is unsatisfiable. It means that if ϕ' is a satisfying valuation of $\mathit{NoBas}(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi)$, then $[\![\exists x''_{a_1}. \cdots . \exists x''_{a_n}. \mathit{Sval}(\phi, \mathrm{pr}^+(\mathcal{P} \parallel \mathcal{Q}), \mathrm{pr}^-(\mathcal{P} \parallel \mathcal{Q}))]\!]_{\phi'}$ is *false*. In other words, $[\![\mathit{Sval}(\phi, \mathrm{pr}^+(\mathcal{P} \parallel \mathcal{Q}), \mathrm{pr}^-(\mathcal{P} \parallel \mathcal{Q}))]\!]_{\phi''}$ is *false* for every $\phi'' \in \{x''_a \mid a \in \mathrm{Im}(\phi)\}$. This, in turn, means that if g is a sortwise function: $\mathrm{Im}(\phi) \to \mathrm{Im}(\phi')$, then $[\![\mathit{Sval}(\phi, \mathrm{pr}^+(\mathcal{P} \parallel \mathcal{Q}), \mathrm{pr}^-(\mathcal{P} \parallel \mathcal{Q}))]\!]_{\phi'_g}$ is *false*. By Item 2 of Lemma 38, it implies that g is not an injection or ϕ' is not a proper $(\emptyset, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of $g(\phi)$. Hence, whenever g is an injection, ϕ' is not strictly smaller than $g(\phi)$ in the $(\emptyset, \mathcal{P} \parallel \mathcal{Q})$-subvaluation order. Since this holds for all satisfying valuations ϕ' of $\mathit{NoBas}(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi)$ and all sortwise injections $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\phi')$, it implies that ϕ is minimal with respect to the $(\emptyset, \mathcal{P} \parallel \mathcal{Q})$-subvaluation order modulo isomorphism.

Finally, we assume that ϕ' is a satisfying valuation of Formula 3. It means that there is a valuation $\phi'' \in \{x''_a \mid a \in \mathrm{Im}(\phi)\}$ such that $[\![\mathit{Sval}(\phi, \mathrm{pr}^+(\mathcal{P} \parallel \mathcal{Q}), \mathrm{pr}^-(\mathcal{P} \parallel \mathcal{Q}))]\!]_{\phi''}$ is *true*. In other words, there is a sortwise function $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\phi')$ such that $[\![\mathit{Sval}(\phi, \mathrm{pr}^+(\mathcal{P} \parallel \mathcal{Q}), \mathrm{pr}^-(\mathcal{P} \parallel \mathcal{Q}))]\!]_{\phi'_g}$ is *true*. By Item 2 of Lemma 38, it implies that g is an injection and ϕ' is a proper $(\emptyset, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of $g(\phi)$. Hence, ϕ' is strictly smaller than ϕ with respect to the $(\emptyset, \mathcal{P} \parallel \mathcal{Q})$-subvaluation order modulo isomorphism. □

The iterative application of Propositions 37 and 39 results in Algorithm 1 that allows us to determine a cut-off set for parameterised systems with a finite basis by starting from the empty set and extending the set with minimal valuations until Formula 1 becomes unsatisfiable for all branch formulae, i.e., all minimal valuations are computed. In each iteration, we first compute some valuation satisfying the topology formula and a branch formula. Then, we minimise the valuation, remove the branch formula specific variables, and add the minimised valuation to the cut-off set meaning that in the following iterations, we do not consider valuations that are greater than the newly computed minimal one. This is repeated until no satisfying valuation is found. Even though the algorithm involves non-determinism in the form of decisions made by the SMT solver, the returned cut-off set is the minimal one and hence unique. That is because the decisions the solver makes do not affect the returned cut-off set, only the order in which the minimal valuations are appended to the set.

In the optimal algorithm, the set is extended valuation by valuation whereas in the original dynamic cut-off algorithm [1], the set is extended in larger chunks. In each iteration of the original algorithm, a sort is queried from an oracle, the cut-off size of the sort is incremented, and all valuations up to the cut-offs are appended to the set until Formula 1 becomes unsatisfiable for all branch formulae. Hence, the optimal algorithm requires at least three solver calls for each valuation in the cut-off set, whereas the original algorithm requires a single solver call for each appended chunk of valuations. The original algorithm is sometimes faster due to fewer calls to the SMT solver (System 9 in Table 1) but in general, it makes more complex solver calls and produces larger cut-off sets than the optimal algorithm presented here, because the chunks may contain unnecessary valuations (Systems 1–3 in Table 1). Moreover, the efficiency of the original algorithm heavily depends on the implementation of the heuristic oracle which is not an issue in the optimal algorithm.

The correctness of Algorithm 1 is stated in Theorem 43, which gives a sufficient condition for termination and states that upon termination, Algorithm 1 indeed returns the optimal cut-off set. For the proof of partial correctness, we need Lemma 40 which states that the minimal basis is obtained as the set of minimal valuations that satisfy the topology formula and a branch formula. In order to show that Algorithm 1 terminates for topologies expressible in the $\exists^*\forall^*$ fragment of FOL, we need Lemma 42 which states that the fragment is decidable and parameterised systems with such topologies have a finite basis. The algorithm also enables us to consider parameterised systems with a finite basis beyond this fragment, but termination depends on the capabilities of the used SMT solver. The solver should be able to decide the satisfiability of Formulae 1–3 used as the loop conditions and produce a satisfying valuation if such exists.

Lemma 40. *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$ be a parameterised system. For each branch formula $\mathcal{B} \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q})$, let $\Phi^{min}_{\mathcal{B}}$ be the set of all minimal valuations (w.r.t. the $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation order) satisfying $\mathcal{F} \wedge \mathcal{B}$ modulo isomorphism. Then the set Φ^{min} of all non-isomorphic valuations $\phi|_{\mathrm{sig}(\mathcal{P} \parallel \mathcal{Q}) \cup \mathrm{sig}(\mathcal{F})}$ such that $\phi \in \Phi^{min}_{\mathcal{B}}$ for some $\mathcal{B} \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q})$ is the minimal basis for $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$.*

input : a parameterised system $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$
output: the optimal cut-off set Φ for $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$

```
// Initialise the cut-off set
Φ ← ∅;
// Loop through all system components
foreach branch formula B ∈ str(P‖Q) do
    // Loop as long as there is a parameter assignment not covered by the cut-off
       set
    while there is a valuation φ ∈ va(F ∧ B | P, Q) satisfying NoBas(P, Q, F, B, Φ) do
        // Minimise the parameter assignment w.r.t. to the size of the sorts
        while there is a valuation φ' ∈ va(F ∧ B | P, Q) satisfying SMin(P, Q, F, B, Φ, φ) do φ ← φ';
        // Minimise the parameter assignment w.r.t. to the values of the predicates
        while there is a valuation φ' ∈ va(F ∧ B | P, Q) satisfying PMin(P, Q, F, B, Φ, φ) do φ ← φ';
        // Remove extra parameters and add the parameter assignment to the cut-off
           set
        Φ ← Φ ∪ {φ|_{sig(P‖Q)∪sig(F)}};
    end
end
// Return the optimal cut-off set
return Φ;
```

Algorithm 1: Optimal cut-off algorithm

Proof. First, we prove that Φ^{min} is a basis of $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$. Let $\mathcal{B} \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q})$ and $\psi \in \mathrm{va}(\mathcal{F} \wedge \mathcal{B} \mid \mathcal{P}, \mathcal{Q})$. Then there is a minimal $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation ϕ of ψ satisfying $\mathcal{F} \wedge \mathcal{B}$. Since ϕ is minimal, there is $\phi' \in \Phi^{min}_{\mathcal{B}}$ and a sortwise injection $g' : \mathrm{Im}(\phi') \to \mathrm{Im}(\phi)$ such that $g'(\phi') = \phi$. This means that $g'(\phi')$ is a $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of ψ satisfying $\mathcal{F} \wedge \mathcal{B}$. By Lemma 35, Φ^{min} is a basis of $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$.

Next, we prove the minimality of Φ^{min} by showing that every basis contains Φ^{min} (modulo isomorphism). Let Φ be a basis of $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$, $\mathcal{B} \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q})$, and $\psi \in \Phi^{min}_{\mathcal{B}}$ a valuation satisfying $\mathcal{F} \wedge \mathcal{B}$. Since Φ is a basis, by Lemma 35, there is a valuation $\phi \in \Phi_{\mathcal{B}}$ and a sortwise injection $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\psi)$ such that $g(\phi)$ is a $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of ψ satisfying $\mathcal{F} \wedge \mathcal{B}$. However, since $\Phi^{min}_{\mathcal{B}}$ is the set of all minimal valuations satisfying $\mathcal{F} \wedge \mathcal{B}$ modulo isomorphism, it means that $\psi = g(\phi)$. Hence, ψ is isomorphic to a valuation in $\Phi_{\mathcal{B}}$, which implies that Φ^{min} is contained in Φ (modulo isomorphism). □

Example 41. Recall the generalised Raft example and the branch formulae $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ of $(\mathit{Raft}' \parallel \mathit{Spec})$ from Example 34. The minimal valuations satisfying $\mathcal{B}_1$ are

1. $\phi'_1 : T_S \mapsto \{s_1\}, T_T \mapsto \{t_1\}, Q_S \mapsto \{(s_1, t_1, s_1)\}, b'_0 \mapsto \mathit{false}, b'_1 \mapsto \mathit{false}, x'_0 \mapsto s_1, x'_1 \mapsto s_1, y' \mapsto t_1$; and
2. $\phi'_2 : T_S \mapsto \{s_1, s_2\}, T_T \mapsto \{t_1\}, Q_S \mapsto \{(s_1, t_1, s_2)\}, b'_0 \mapsto \mathit{false}, b'_1 \mapsto \mathit{false}, x'_0 \mapsto s_1, x'_1 \mapsto s_2, y' \mapsto t_1$.

For the branch formula $\mathcal{B}_2$, the minimal satisfying valuations are

3. $\phi'_3 : T_S \mapsto \{s_1, s_2\}, T_T \mapsto \{t_1\}, Q_S \mapsto \emptyset, b'_0 \mapsto \mathit{true}, b'_1 \mapsto \mathit{false}, x'_0 \mapsto s_1, x'_1 \mapsto s_1, x'_2 \mapsto s_2, y' \mapsto t_1$;
4. $\phi'_4 : T_S \mapsto \{s_1, s_2\}, T_T \mapsto \{t_1\}, Q_S \mapsto \emptyset, b'_0 \mapsto \mathit{true}, b'_1 \mapsto \mathit{false}, x'_0 \mapsto s_2, x'_1 \mapsto s_1, x'_2 \mapsto s_2, y' \mapsto t_1$; and
5. $\phi'_5 : T_S \mapsto \{s_1, s_2, s_3\}, T_T \mapsto \{t_1\}, Q_S \mapsto \emptyset, b'_0 \mapsto \mathit{true}, b'_1 \mapsto \mathit{false}, x'_0 \mapsto s_1, x'_1 \mapsto s_2, x'_2 \mapsto s_3, y' \mapsto t_1$,

and the minimal satisfying valuations of the branch formula $\mathcal{B}_3$ are

6. $\phi'_6 : T_S \mapsto \{s_1\}, T_T \mapsto \{t_1\}, Q_S \mapsto \{(s_1, t_1, s_1)\}, b'_1 \mapsto \mathit{true}, x'_0 \mapsto s_1, x'_1 \mapsto s_1, x'_2 \mapsto s_1, y' \mapsto t_1$;
7. $\phi'_7 : T_S \mapsto \{s_1, s_2\}, T_T \mapsto \{t_1\}, Q_S \mapsto \{(s_1, t_1, s_2)\}, b'_1 \mapsto \mathit{true}, x'_0 \mapsto s_1, x'_1 \mapsto s_1, x'_2 \mapsto s_2, y' \mapsto t_1$;

8. $\phi'_8 : T_S \mapsto \{s_1, s_2\}, T_T \mapsto \{t_1\}, Q_S \mapsto \{(s_1, t_1, s_2), (s_2, t_1, s_2)\}, b'_1 \mapsto \mathit{true}, x'_0 \mapsto s_1, x'_1 \mapsto s_2, x'_2 \mapsto s_2, y' \mapsto t_1$;
9. $\phi'_9 : T_S \mapsto \{s_1, s_2\}, T_T \mapsto \{t_1\}, Q_S \mapsto \{(s_1, t_1, s_2), (s_2, t_1, s_2)\}, b'_1 \mapsto \mathit{true}, x'_0 \mapsto s_2, x'_1 \mapsto s_1, x'_2 \mapsto s_2, y' \mapsto t_1$; and
10. $\phi'_{10} : T_S \mapsto \{s_1, s_2, s_3\}, T_T \mapsto \{t_1\}, Q_S \mapsto \{(s_1, t_1, s_3), (s_2, t_1, s_3)\}, b'_1 \mapsto \mathit{true}, x'_0 \mapsto s_1, x'_1 \mapsto s_2, x'_2 \mapsto s_3, y' \mapsto t_1$.

When we remove the primed variables, we are left with six non-isomorphic valuations that are precisely the valuations $\phi_1, \phi_2, \phi_3, \phi_4, \phi_5, \phi_6$ of Example 20. Hence, by Lemma 40, we have proved that the set $\{\phi_1, \phi_2, \phi_3, \phi_4, \phi_5, \phi_6\}$ is the optimal cut-off set for our generalised Raft model. □

Lemma 42. *Let $\mathcal{F}$ be a formula in the $\exists^*\forall^*$ fragment of FOL. Then the satisfiability of $\mathcal{F}$ (over finite valuations) is decidable and for any implementation PLTS $\mathcal{P}$ and specification PLTS $\mathcal{Q}$, the parameterised system $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$ has a finite basis.*

Proof. The decidability of the $\exists^*\forall^*$ fragment over finite valuations is well-known [19, 36].

Let us then consider a topology $\mathcal{F} := \exists x_1. \cdots . \exists x_n . \mathcal{U}$, where $\mathcal{U}$ is in the $\forall^*$ fragment. Without loss of generality, we may assume that the variables $x_1, \ldots, x_n$ do not occur in $\mathcal{P}$ and $\mathcal{Q}$. By Theorem 50 in [17], we know that the parameterised system $(\mathcal{P}, \mathcal{Q}, \mathcal{U})$ has a finite basis Φ'.

Next, we will show that $\Phi := \{\phi'|_{\mathrm{dom}(\phi')\setminus\{x_1,\ldots,x_n\}} \mid \phi' \in \Phi'\}$ is a finite basis of the parameterised system $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$. Let $\psi \in \mathrm{va}(\mathcal{F} \mid \mathcal{P}, \mathcal{Q})$ and $w \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, \psi)$. Then there is $\psi' \in \mathrm{ext}(\psi, \{x_1, \ldots, x_n\})$ such that $\psi' \in \mathrm{va}(\mathcal{U} \mid \mathcal{P}, \mathcal{Q})$ and $w \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, \psi')$. Since Φ' is the basis of $(\mathcal{P}, \mathcal{Q}, \mathcal{U})$, there is a valuation $\phi' \in \Phi'$ and a sortwise injection g such that $w \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, g(\phi'))$ and $g(\phi')$ is a $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of ψ'. Because $x_1, \ldots, x_n$ do not occur in $\mathcal{P} \parallel \mathcal{Q}$, it implies that $\phi := \phi'|_{\mathrm{dom}(\phi')\setminus\{x_1,\ldots,x_n\}}$ is a valuation in Φ such that $w \in \mathrm{str}(\mathcal{P} \parallel \mathcal{Q}, g(\phi))$ and $g(\phi)$ is a $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation of ψ. Hence, Φ is a finite basis of the parameterised system $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$. □

Theorem 43 (Correctness of Optimal cut-off algorithm). *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$ be a parameterised system.*

1. *If Algorithm 1 terminates with a set Φ of valuations, then Φ is a cut-off set for $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$. Moreover, Φ is optimal in the sense that it is the minimal finite basis of $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$.*
2. *If the parameterised system $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$ has a finite basis and the satisfiability (over finite valuations) of the formulas of the form $\mathcal{F} \wedge \mathcal{U}$ (where $\mathcal{U}$ is in the $\exists^*\forall^*$ fragment) is decidable, then Algorithm 1 terminates.*
3. *If $\mathcal{F}$ is in the $\exists^*\forall^*$ fragment, then Algorithm 1 terminates.*

Proof.

1. Let us assume that Algorithm 1 terminates with a set Φ. Let $\mathcal{B}_1, \ldots, \mathcal{B}_n$ be the branch formulae in $\mathrm{str}(\mathcal{P} \| \mathcal{Q})$ and let us assume that this is the order in which they are processed in the for-each loop. Then for every $i \in \{1, \ldots, n\}$ there is a set $\Phi_i \subseteq \mathrm{va}(\mathcal{F} \mid \mathcal{P}, \mathcal{Q})$ of valuations such that $\mathit{NoBas}(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}_i, \Phi_i)$ is unsatisfiable. Since $\Phi_1 \subseteq \Phi_2 \subseteq \cdots \subseteq \Phi_n = \Phi$, it means that $\mathit{NoBas}(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}_i, \Phi)$ is unsatisfiable for all $i \in \{1, \ldots, n\}$. By Item 1 of Proposition 37, it implies that Φ is a finite basis of $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$, and by Proposition 31, we see that Φ is a cut-off set for $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$.

   In order to prove that the computed Φ is the minimal finite basis of $(\mathcal{P}, \mathcal{Q}, \mathcal{F})$, we show that Φ is a subset of Φ^{min} during the execution of the algorithm. Let us assume that the computed cut-off set Φ is $\{\phi_1|_\Sigma, \ldots, \phi_n|_\Sigma\}$, where $\Sigma = \mathrm{sig}(\mathcal{P} \parallel \mathcal{Q}) \cup \mathrm{sig}(\mathcal{F})$ and ϕ_i denotes the value of ϕ in the end of the i^{th} iteration of the outer while loop. We claim that for each $i \in \{0, 1, \ldots, n\}$, $\Phi^i := \{\phi_1|_\Sigma, \ldots, \phi_i|_\Sigma\} \subseteq \Phi^{min}$ and $\phi_1|_\Sigma, \ldots, \phi_i|_\Sigma$ are non-isomorphic. Obviously, $\Phi^0 = \emptyset \subseteq \Phi^{min}$ and the valuations in Φ^0 are non-isomorphic. Let us then assume that $\Phi^i \subseteq \Phi^{min}$ for some $i \in \{0, 1, \ldots, n\}$ and that the valuations $\phi_1|_\Sigma, \ldots, \phi_i|_\Sigma$ are non-isomorphic. We need to show that ϕ_{i+1} is a minimal valuation in

$\mathrm{va}(\mathcal{F} \wedge \mathcal{B} \mid \mathcal{P}, \mathcal{Q})$ for some $\mathcal{B} \in \mathrm{str}(\mathcal{P} \| \mathcal{Q})$ and $\phi_{i+1}|_\Sigma$ is non-isomorphic to the valuations in Φ^i. For that purpose, let us assume that ϕ'_{i+1} is the valuation that makes the condition $SMin(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi^i, \phi'_{i+1})$ of the first inner while loop unsatisfiable in the i^{th} iteration of the outer while loop. By Item 1 of Proposition 39, we know that ϕ'_{i+1} is a minimal valuation with respect to the $(\mathbb{T}, \emptyset, \emptyset)$-subvaluation order (modulo isomorphism) satisfying $NoBas(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi^i)$. By Item 2 of Proposition 39, we also know that ϕ_{i+1} is a minimal valuation with respect to the $(\emptyset, \mathcal{P} \parallel \mathcal{Q})$-subvaluation order (modulo isomorphism) satisfying $NoBas(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi^i)$. Especially, ϕ_{i+1} is smaller (but not necessarily strictly) than ϕ'_{i+1} with respect to the $(\emptyset, \mathcal{P} \parallel \mathcal{Q})$-subvaluation order (modulo isomorphism). This implies that ϕ_{i+1} is a minimal valuation with respect to the $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation order (modulo isomorphism) satisfying $NoBas(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi^i)$. By Item 2 of Proposition 37, ϕ_{i+1} is a minimal valuation with respect to the $(\mathbb{T}, \mathcal{P} \parallel \mathcal{Q})$-subvaluation order (modulo isomorphism) satisfying $\mathcal{F} \wedge \mathcal{B}$ such that for every valuation $\phi \in \Phi^i_{\mathcal{B}}$ and for every sortwise injection $g : \mathrm{Im}(\phi) \to \mathrm{Im}(\phi_{i+1})$, $g(\phi)$ is not a $(\mathbb{T}, \mathcal{P} \| \mathcal{Q})$-subvaluation of ϕ_{i+1}. This implies that $\phi_{i+1}|_\Sigma$ must be non-isomorphic to the valuations in Φ^i. By the induction principle, this implies that $\Phi = \Phi^n \subseteq \Phi^{min}$. Since Φ is a finite basis and contained in Φ^{min}, which, by Lemma 40, is the minimal basis, this means that Φ must be the minimal finite basis Φ^{min}.

2. In order to show the termination of the algorithm, we need to prove a convergent for each while loop and show that the conditions of the while loops are decidable (over finite valuations).

   Let us first consider the decidability of the loop conditions. The formula $NoBas(\mathcal{P}, \mathcal{Q}, \mathcal{F}, \mathcal{B}, \Phi)$ of the outer while loop consists of several conjuncts: the topology formula $\mathcal{F}$, a branch formula $\mathcal{B}$ involving only conjunctions and existential quantification, and universally quantified conjuncts. Since quantification (over non-empty sets) can be pushed outside the conjunctions, the condition can be put to the form $\mathcal{F} \wedge \mathcal{U}$, where $\mathcal{U}$ is in the $\exists^*\forall^*$ fragment. The formulae of the inner while loops involve an additional existentially quantified conjunct of quantifier-free and universally quantified conjuncts, but also in these cases, quantification is over non-empty sets and can be pushed outside the conjunctions yielding to a formula of the form $\mathcal{F} \wedge \mathcal{U}$ with $\mathcal{U}$ being in the $\exists^*\forall^*$ fragment. Hence, by the assumption, the conditions of the while loops are decidable (over finite valuations).

   Let us then prove a convergent for each while loop. If the parameterised system has a finite basis, then also the minimal basis is finite. By the proof of Item 1, it implies that the size of $\Phi^{min} \setminus \Phi$, where the set difference is modulo isomorphism, decreases in each iteration of the outer while loop. By Proposition 39, the size of the image of the sorts, $|\mathrm{Im}(\phi)|$, decreases in each iteration of the first inner while loop and the size of the values of the predicates occurring under an even number of negations plus the size of the complement values of the predicates occurring under an odd number of negations, $\sum_{F \in \mathrm{pr}^+(\mathcal{P} \| \mathcal{Q})} |\phi(F)| + \sum_{F \in \mathrm{pr}^-(\mathcal{P} \| \mathcal{Q})} |\overline{\phi(F)}|$, decreases in each iteration of the second inner while loop. Hence, the algorithm terminates.

3. Follows from Lemma 42 and (2). □

We have implemented Algorithm 1 in the Bounds tool [39]. The tool takes a parameterised system as an input expressed in a language resembling machine readable CSP [3], a glimpse of the input language can be found in Appendix A describing our Raft example. After that, the tool computes a cut-off set by using the optimal cut-off algorithm, and provided the algorithm terminates, produces an LTS refinement checking task for each valuation in the cut-off set. Finally, the verification is completed by refinement checking the instances. Bounds consists of 30 kLOC of C/C++ code, roughly half of which is autogenerated by the ANTLR parser generator [40]. Additionally, the tool uses Z3 SMT solver [41] for testing satisfiability and generating satisfying valuations, the nauty package [42] for detecting isomorphic valuations, and FDR [43] for finite-state refinement checking. The whole process is fully automated; the user only has to provide the description of a parameterised system and wait for termination.

If the parameterised system is not correct, the cut-off set is computed as usual, but there is an instance of the system generated by a valuation in the cut-off set for which refinement checking fails. In this case, the tool reports that the parameterised system is not correct and returns a counterexample. Otherwise, if the parameterised system is correct, then all LTS refinement checks are successful and the tool simply reports

the parameterised system to be correct. This also means that 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 PLTS formalism is compositional. Bounds is publicly available at [44].

Example 44. We have applied Bounds to several system models by using the optimal, original dynamic [1], and static cut-off algorithms [17, 16] (Table 1). The (non-generalised) Raft model for the static algorithm uses specific *quorum function variables* [16] which in our formalism are modelled in FOL. The topology of the generalised Raft models cannot be modelled by using quorum functions nor without existential quantification. Therefore, these models are outside the scope of the static algorithms. The tree topology of taDOM2+ and the lower bound for the number of transactions are naturally modelled by using existential quantification but for the static algorithm, which does not support it, they are modelled by using free variables. Otherwise the models are identical.

In general, we can see that larger cut-off sets take longer to compute. The size of the cut-off set depends not only on the number of the parameters but also on the structure of a parameterised system. If the parameterised system has many variables as parameters and involves lots of nested replicated parallel compositions, this usually implies big cut-off sizes for the sorts and, hence, a large cut-off set.

Obviously, the size of the cut-off set affects refinement checking time as well. However, the refinement checking time also depends on the number and size of elementary PLTSs in the parameterised system as well as the number of times they are instantiated. This means that a high number of (replicated) parallel compositions and large elementary PLTSs predict long refinement checking time.

In the case of Systems 5–9, the topology of which is within the $\forall^*$ fragment, all three algorithms are equal in terms of cut-offs. There is no significant difference in the running time of the algorithms either except in the last case where the computed cut-off set is the largest and consequently, the optimal algorithm is the slowest due to the highest number of calls made to the SMT solver.

In the case of Systems 1–4, the topology of which is outside the $\forall^*$ fragment, the optimal algorithm outperforms the other algorithms both in terms of the running time and the size of the cut-off sets. Moreover, the optimal algorithm is the only one terminating on all cases. Hence, the optimal cut-off algorithm not only enables extending the application domain of the static ones but also provides more compact cut-offs, which is important in order to keep the refinement checking part feasible.

All experiments were made on a quad-core (octa-thread) Intel i7-4790 with 16 GB of memory running Ubuntu 18.04.2 LTS by using Z3 4.8.4 as a back-end SMT solver and FDR 4.2.3 as a back-end refinement checker. An example run of Bounds on the generalised Raft model is in Appendix B. □

7. Conclusions and Future Work

We have shown how to parameterise the calculus of LTSs by using first order constructs, sorts, variables, and predicates, in a compositionality preserving way. As the main contribution, we have provided a semi-algorithm for reducing a refinement checking task in the parameterised LTS formalism to a finite set of refinement checks between LTSs. This is done by iteratively computing a cut-off set of parameter values such that in order to prove a parameterised system implementation correct with respect to its specification, it is sufficient to consider only finitely many instances generated by the parameter values in the cut-off set. The algorithm not only combines existing static cut-off techniques but also extends their application domain beyond known decidable fragments. The algorithm is implemented in a tool and applied to several system models, including the leader election phase of the generalised (Byzantine) Raft protocol.

The original version of the algorithm is presented in the conference version of this work [1]. The downside of the original algorithm is that it is not guaranteed to produce an optimal cut-off set and its performance depends on the implementation of a heuristic oracle. The new version of the algorithm presented here overcomes these limitations; it does not involve an oracle and always produces the optimal cut-off set. By using the new optimal algorithm, we were able to analyse the leader election phase of the Byzantine Raft which turned out to be an infeasible task for the original algorithm. To the best of our knowledge, this is the first time a Byzantine variant of the Raft leader election is automatically verified.

Table 1: The performance of the optimal cut-off algorithm with respect to the original dynamic and static ones, $|T|$ is the cut-off size for a sort T, $|\Phi|$ is the size of the cut-off set, and t is the time (in seconds) taken by the computation of the cut-off set Φ plus refinement checking the ϕ-instances for all $\phi \in \Phi$.

System	**Parameters**	**Optimal**		**Original dynamic [1]**		**Static [17, 16]**	
		cut-offs	**t(s)**	**cut-offs**	**t(s)**	**cut-offs**	**t(s)**
1. Raft [16]	servers (S), terms (T), quorum topology	$\|\mathbf{S}\| = \mathbf{3}$, $\|\mathbf{T}\| = \mathbf{1}$, $\|\mathbf{\Phi}\| = \mathbf{7}$	**1+1**	$\|S\| = 3$, $\|T\| = 1$, $\|\Phi\| = 20$	7+2	$\|S\| = 7$, $\|T\| = 1$, $\|\Phi\| > 10^5$	out of mem.
2. Generalised Raft	servers (S), terms (T), quorum topology	$\|\mathbf{S}\| = \mathbf{3}$, $\|\mathbf{T}\| = \mathbf{1}$, $\|\mathbf{\Phi}\| = \mathbf{6}$	**1+1**	$\|S\| = 3$, $\|T\| = 1$, $\|\Phi\| = 74$	1+8	n/a	n/a
3. Generalised Byzantine Raft	servers (S), terms (T), Byzantine quorum topology	$\|\mathbf{S}\| = \mathbf{4}$, $\|\mathbf{T}\| = \mathbf{1}$, $\|\mathbf{\Phi}\| = \mathbf{13}$	**1+1**	$\|S\| \geq 4$, $\|T\| \geq 1$, $\|\Phi\| =?$	out of mem.	n/a	n/a
4. taDOM2+	2+ transactions (T), nodes (N), tree topology	$\|\mathbf{T}\| = \mathbf{2}$, $\|\mathbf{N}\| = \mathbf{3}$, $\|\mathbf{\Phi}\| = \mathbf{7}$	**1+ 1116**	$\|\mathbf{T}\| = \mathbf{2}$, $\|\mathbf{N}\| = \mathbf{3}$, $\|\mathbf{\Phi}\| = \mathbf{7}$	**1+ 1121**	$\|T\| = 4$, $\|N\| = 4$, $\|\Phi\| = 45$	11+ out of mem.
5. taDOM2+ [17]	transactions (T), nodes (N), forest topology	$\|T\| = 2$, $\|N\| = 3$, $\|\Phi\| = 14$	1+ 1125	$\|T\| = 2$, $\|N\| = 3$, $\|\Phi\| = 14$	1+ 1122	$\|T\| = 2$, $\|N\| = 3$, $\|\Phi\| = 14$	1+ 1130
6. Shared resources [17]	users (U), resources (R), forest topology	$\|U\| = 2$, $\|R\| = 3$, $\|\Phi\| = 6$	1+6	$\|U\| = 2$, $\|R\| = 3$, $\|\Phi\| = 6$	1+6	$\|U\| = 2$, $\|R\| = 3$, $\|\Phi\| = 6$	1+6
7. Shared resources [17]	users (U), resources (R), ring topology	$\|U\| = 4$, $\|R\| = 1$, $\|\Phi\| = 4$	1+1	$\|U\| = 4$, $\|R\| = 1$, $\|\Phi\| = 4$	1+1	$\|U\| = 4$, $\|R\| = 1$, $\|\Phi\| = 4$	1+1
8. Token ring [39]	users (U), ring topology	$\|U\| = 4$, $\|\Phi\| = 3$	1+1	$\|U\| = 4$, $\|\Phi\| = 3$	1+1	$\|U\| = 4$, $\|\Phi\| = 3$	1+1
9. Ring with 2 tokens [17]	users (U), ring topology	$\|U\| = 5$, $\|\Phi\| = 30$	156+ 7	$\|U\| = 5$, $\|\Phi\| = 30$	**10+ 7**	$\|U\| = 5$, $\|\Phi\| = 30$	25+ 7

In future, we aim to further extend the algorithm to other process algebraic formalisms such as modal interface automata [18]. In general, the algorithm should be applicable to formalisms satisfying Proposition 5. However, extending the technique to other modelling formalisms is probably difficult, for example, it is unlikely that Promela [34] models could be easily treated with our approach because Promela does not have the idempotence property. We are also investigating ways to integrate our optimal cut-off algorithm with the induction method [22] and data-independence results [23] as in [45, 46]. Since the behaviour of Raft is independent of the type of stored data, this could enable us to consider the log replication phase of (Byzantine) Raft as well, i.e., to analyse the correctness of the full Raft protocol completely automatically.

Acknowledgement

The research is partly funded by Academy of Finland projects 313469 and 277522.

References

[1] A. Siirtola, K. Heljanko, Dynamic cut-off algorithm for parameterised refinement checking, in: K. Bae, P. C. Ölveczky (Eds.), Formal Aspects of Component Software, volume 11222 of *LNCS*, Springer, 2018, pp. 256–276.

[2] C. A. R. Hoare, Communicating Sequential Processes, Prentice-Hall, 1985.

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

[4] E. A. Emerson, V. Kahlon, Reducing model checking of the many to the few, in: D. A. McAllester (Ed.), CADE-17, volume 1831 of *LNCS*, Springer, 2000, pp. 236–254.

[5] E. A. Emerson, K. S. Namjoshi, On reasoning about rings, Int. J. Found. Comput. Sci. 14 (2003) 527–550.

[6] E. A. Emerson, V. Kahlon, Model checking large-scale and parameterized resource allocation systems, in: J.-P. Katoen, P. Stevens (Eds.), TACAS '02, volume 2280 of *LNCS*, Springer, 2002, pp. 251–265.

[7] E. A. Emerson, V. Kahlon, Exact and efficient verification of parameterized cache coherence protocols, in: D. Geist, E. Tronci (Eds.), CHARME '03, volume 2860 of *LNCS*, Springer, 2003, pp. 247–262.

[8] E. A. Emerson, V. Kahlon, Parameterized model checking of ring-based message passing systems, in: J. Marcinkowski, A. Tarlecki (Eds.), CSL '04, volume 3210 of *LNCS*, Springer, 2004, pp. 325–339.

[9] A. Bouajjani, P. Habermehl, T. Vojnar, Verification of parametric concurrent systems with prioritised FIFO resource management, Form. Method. Syst. Des. 32 (2008) 129–172.

[10] A. Kaiser, D. Kroening, T. Wahl, Dynamic cutoff detection in parameterized concurrent programs, in: T. Touili, B. Cook, P. Jackson (Eds.), CAV '10, volume 6174 of *LNCS*, Springer, 2010, pp. 645–659.

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

[12] P. Abdulla, F. Haziza, L. Holík, Parameterized verification through view abstraction, Int. J. Softw. Tools Technol. Transf. 18 (2016) 495–516.

[13] B. Aminof, T. Kotek, S. Rubin, F. Spegni, H. Veith, Parameterized model checking of rendezvous systems, Distrib. Comput. 31 (2018) 187–222.

[14] M. Haustein, T. Härder, Optimizing lock protocols for native XML processing, Data Knowl. Eng. 65 (2008) 147–173.

[15] D. Ongaro, J. Ousterhout, In search of an understandable consensus algorithm, in: G. Gibson, N. Zeldovich (Eds.), USENIX ATC '14, USENIX Association, 2014, pp. 305–320.

[16] A. Siirtola, Refinement checking parameterised quorum systems, in: A. Legay, K. Schneider (Eds.), ACSD '17, IEEE, 2017, pp. 39–48.

[17] A. Siirtola, J. Kortelainen, Multi-parameterised compositional verification of safety properties, Inform. Comput. 244 (2015) 23–48.

[18] A. Siirtola, K. Heljanko, Parametrised modal interface automata, ACM Trans. Embed. Comput. Syst. 14 (2015) 65:1–65:25.

[19] Y. Gurevich, On the classical decision problem, in: G. Rozenberg, A. Salomaa (Eds.), Current Trends In Theoretical Computer Science: Essays and Tutorials, volume 40 of *World Scientific Series in Computer Science*, World Scientific, 1993, pp. 254–265.

[20] O. Marić, C. Sprenger, D. Basin, Cutoff bounds for consensus algorithms, in: R. Majumdar, V. Kunčak (Eds.), CAV '17, volume 10427 of *LNCS*, Springer, 2017, pp. 217–237.

[21] C. Copeland, H. Zhong, Tangaroa: A Byzantine fault tolerant Raft, 2014. URL: `http://www.scs.stanford.edu/14au-cs244b/labs/projects/copeland_zhong.pdf`.

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

[23] R. S. Lazić, A Semantic Study of Data Independence with Applications to Model Checking, Ph.D. thesis, Oxford University, 1999.

[24] S. J. Creese, Data Independent Induction: CSP Model Checking of Arbitrary Sized Networks, Ph.D. thesis, Oxford University, 2001.

[25] R. P. Kurshan, K. L. McMillan, A structural induction theorem for processes, Inform. Comput. 117 (1995) 1–11.

[26] P. Wolper, V. Lovinfosse, Verifying properties of large sets of processes with network invariants, in: J. Sifakis (Ed.), Automatic Verification Methods for Finite State Systems '89, volume 407 of *LNCS*, Springer, 1990, pp. 68–80.
[27] R. S. Lazić, D. Nowak, A unifying approach to data-independence, in: C. Palamidessi (Ed.), CONCUR '00, volume 1877 of *LNCS*, Springer, 2000, pp. 581–595.
[28] Y. Hanna, D. Samuelson, S. Basu, H. Rajan, Automating cut-off for multi-parameterized systems, in: J. S. Dong, H. Zhu (Eds.), ICFEM '10, volume 6447 of *LNCS*, Springer, 2010, pp. 338–354.
[29] E. Clarke, M. Talupur, T. Touili, H. Veith, Verification by network decomposition, in: P. Gardner, N. Yoshida (Eds.), CONCUR '04, volume 3170 of *LNCS*, Springer, 2004, pp. 276–291.
[30] P. Liu, T. Wahl, CUBA: interprocedural Context-UnBounded Analysis of concurrent programs, in: J. S. Foster, D. Grossman (Eds.), PLDI '18, ACM, 2018, pp. 105–119.
[31] L. Zuck, A. Pnueli, Model checking and abstraction to the aid of parameterized systems (a survey), Comput. Lang. Syst. Str. 30 (2004) 139–169.
[32] A. Finkel, P. Schnoebelen, Well-structured transition systems everywhere!, Theor. Comput. Sci. 256 (2001) 63–92.
[33] D. Woos, J. R. Wilcox, S. Anton, Z. Tatlock, M. D. Ernst, T. Anderson, Planning for change in a formal verification of the Raft consensus protocol, in: J. Avigad, A. Chlipala (Eds.), CPP '16, ACM, 2016, pp. 154–165.
[34] G. J. Holzmann, The SPIN model checker: Primer and reference manual, Addison-Wesley Reading, 2004.
[35] J. H. Gallier, Logic for Computer Science: Foundations of Automatic Theorem Proving, Courier Dover Publications, 2015.
[36] A. Abadi, A. Rabinovich, M. Sagiv, Decidable fragments of many-sorted logic, J. Symb. Comput. 45 (2010) 153–172.
[37] C. J. Colbourn, J. H. Dinitz, D. R. Stinson, Quorum systems constructed from combinatorial designs, Inf. Comput. 169 (2001) 160–173.
[38] A. Siirtola, Automated multiparameterised verification by cut-offs, in: ICFEM '10, volume 6447 of *LNCS*, Springer, 2010, pp. 321–337.
[39] A. Siirtola, Bounds2: A tool for compositional multi-parametrised verification, in: E. Ábrahám, K. Havelund (Eds.), TACAS '14, volume 8413 of *LNCS*, Springer, 2014, pp. 599–604.
[40] J. Bovet, T. Parr, ANTLRWorks: an ANTLR grammar development environment, Software Pract. Exper. 38 (2008) 1305–1332.
[41] L. De Moura, N. Bjørner, Z3: An efficient SMT solver, in: C. R. Ramakrishnan, J. Rehof (Eds.), TACAS '08, volume 4963 of *LNCS*, Springer, 2008, pp. 337–340.
[42] B. D. McKay, A. Piperno, Practical graph isomorphism II, J. Symb. Comput. 60 (2014) 94 – 112.
[43] T. Gibson-Robinson, P. Armstrong, A. Boulgakov, A. W. Roscoe, FDR3: A parallel refinement checker for CSP, STTT 18 (2016) 149–167.
[44] A. Siirtola, Bounds website, 2019. `http://cc.oulu.fi/~asiirtol/bounds`.
[45] A. Siirtola, Cut-offs with network invariants, in: L. Gomez, V. Khomenko, J. Fernandes (Eds.), ACSD '10, IEEE, 2010, pp. 105–114.
[46] A. Siirtola, K. Heljanko, Parametrised compositional verification with multiple process and data types, in: J. Carmona, M. T. Lazarescu, M. Pietkiewicz-Koutny (Eds.), ACSD '13, IEEE, 2013, pp. 67–76.

Appendix A. Bounds Code for Generalised (Byzantine) Raft Leader Election

```
sort S
sort T

pred QS : S,T,S
// Uncomment for Byzantine version
// pred NB : T,S

var x0 : S
var x1 : S
var x2 : S
var x3 : S
var y : T

// Comment for Byzantine version
frml Qrm = \/ x0,x1,y: ((\/ x2: !QS(x0,y,x2)) | (\/ x2: !QS(x1,y,x2)) |
                        !(\/ x2: !(QS(x0,y,x2) & QS(x1,y,x2))))
// Uncomment for Byzantine version
// frml Byz = \/ x0,x1,y: ((\/ x2: !QS(x0,y,x2)) | (\/ x2: !QS(x1,y,x2)) |
//                         !(\/ x2: !(QS(x0,y,x2) & QS(x1,y,x2)  & NB(y,x2))))

chan vote : S, T, S
chan candidate : S, T
```

```
chan leader : S, T

plts Spec2 =
    lts
        I   =       leader(x0,y) -> S0
                []  leader(x1,y) -> S1
        S0  =       leader(x0,y) -> S0
        S1  =       leader(x1,y) -> S1
    from I

plts Spec = (|| x0,x1,x2,y: [QS(x0,y,x2) & QS(x1,y,x2)] Spec2)

plts Ldr2 =
    lts
        C   =       candidate(x0,y) -> C1
                []  vote(x1,y,x0) -> C
        C1  =       vote(x1,y,x0) -> L
        L   =       leader(x0,y) -> L
                []  vote(x1,y,x0) -> L
    from C

plts Flw3 =
    lts
        F   =       candidate(x0,y) -> F0
                []  vote(x0,y,x1) -> F1
                []  vote(x0,y,x2) -> F2
        F1  =       vote(x0,y,x1) -> F1
        F2  =       vote(x0,y,x2) -> F2
        F0  =       vote(x0,y,x0) -> F0
    from F

// Comment for Byzantine version
plts Raft = || x0: ((|| y,x1: [QS(x0,y,x1)] Ldr2) || (|| x1,x2: [!x1=x2] || y: Flw3))
// Uncomment for Byzantine version
// plts BRaft = || x0: ((|| y,x1: [QS(x0,y,x1)] Ldr2) ||
//                      (|| x1,x2: [!x1=x2] || y: [NB(y,x0)] Flw3))

pset LE = (_) x0,x1,y: {candidate(x0,y), vote(x0,y,x1)}

// Comment for Byzantine version
trace refinement: verify Raft \ LE against Spec when Qrm
// Uncomment for Byzantine version
// trace refinement: verify BRaft \ LE against Spec when Byz
```

Appendix B. Run of Bounds on Generalised Raft Leader Election

```
This is Bounds 3.1!
Created by Antti Siirtola 2010-2019 (contact: antti.siirtola@oulu.fi)

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

Computing the optimal cut-off set.
The size of the cut-off set is now 0.

Computing the extended valuations...
```

```
...Done!
Found 0 non-isomorphic valuations.
(#valuations: 0, #canonical forms: 0, max #valuations stored: 0, #isomorphs removed: 0,
#branches pruned: 0)
Searching for satisfying valuations...
...New valuation found:
T -> {T0}
S -> {S0,S1,S2}
QS -> {(S0,T0,S0),(S0,T0,S1),(S0,T0,S2),(S1,T0,S0),(S1,T0,S1),(S1,T0,S2),(S2,T0,S0),(S2,T0,S1),
(S2,T0,S2)}
Minimising the values of sorts...
...New valuation found:
T -> {T0}
S -> {S0,S1}
QS -> {(S0,T0,S0),(S0,T0,S1),(S1,T0,S0),(S1,T0,S1)}
Minimising the values of sorts...
...New valuation found:
T -> {T0}
S -> {S0}
QS -> {(S0,T0,S0)}
Minimising the values of sorts...
...Done!
Minimising the values of predicates...
...Done!
The size of the cut-off set is now 1.

Computing the extended valuations...
...Done!
Found 1 non-isomorphic valuations.
(#valuations: 3, #canonical forms: 3, max #valuations stored: 0, #isomorphs removed: 0,
#branches pruned: 0)
Searching for satisfying valuations...
...New valuation found:
T -> {T0}
S -> {S0,S1,S2}
QS -> {(S2,T0,S0),(S2,T0,S1)}
Minimising the values of sorts...
...New valuation found:
T -> {T0}
S -> {S0,S1}
QS -> {(S1,T0,S0),(S1,T0,S1)}
Minimising the values of sorts...
...Done!
Minimising the values of predicates...
...New valuation found:
T -> {T0}
S -> {S0,S1}
QS -> {(S1,T0,S0)}
Minimising the values of sorts...
...Done!
The size of the cut-off set is now 2.

Computing the extended valuations...
...Done!
Found 1 non-isomorphic valuations.
(#valuations: 7, #canonical forms: 4, max #valuations stored: 2, #isomorphs removed: 0,
```

```
#branches pruned: 0)
Searching for satisfying valuations...
...All found for this component, continuing to the next one.
Computing the extended valuations...
...Done!
Found 4 non-isomorphic valuations.
(#valuations: 19, #canonical forms: 14, max #valuations stored: 4, #isomorphs removed: 0,
#branches pruned: 0)
Searching for satisfying valuations...
...New valuation found:
T -> {T0}
S -> {S0,S1,S2}
QS -> {}
Minimising the values of sorts...
...New valuation found:
T -> {T0}
S -> {S0,S1}
QS -> {}
Minimising the values of sorts...
...Done!
Minimising the values of predicates...
...Done!
The size of the cut-off set is now 3.

Computing the extended valuations...
...Done!
Found 2 non-isomorphic valuations.
(#valuations: 9, #canonical forms: 7, max #valuations stored: 2, #isomorphs removed: 1,
#branches pruned: 0)
Searching for satisfying valuations...
...New valuation found:
T -> {T0}
S -> {S0,S1,S2}
QS -> {}
Minimising the values of sorts...
...Done!
Minimising the values of predicates...
...Done!
The size of the cut-off set is now 4.

Computing the extended valuations...
...Done!
Found 3 non-isomorphic valuations.
(#valuations: 13, #canonical forms: 10, max #valuations stored: 3, #isomorphs removed: 3,
#branches pruned: 0)
Searching for satisfying valuations...
...All found for this component, continuing to the next one.
Computing the extended valuations...
...Done!
Found 2 non-isomorphic valuations.
(#valuations: 55, #canonical forms: 39, max #valuations stored: 11, #isomorphs removed: 5,
#branches pruned: 0)
Searching for satisfying valuations...
...New valuation found:
T -> {T0}
S -> {S0,S1,S2,S3,S4,S5,S6}
```

```
QS -> {(S3,T0,S1),(S3,T0,S2),(S3,T0,S4),(S3,T0,S6),(S5,T0,S0),(S5,T0,S1),(S5,T0,S4),(S5,T0,S6)}
Minimising the values of sorts...
...New valuation found:
T -> {T0}
S -> {S0,S1,S2,S3,S4,S5}
QS -> {(S2,T0,S0),(S2,T0,S1),(S2,T0,S3),(S2,T0,S5),(S3,T0,S0),(S3,T0,S2),(S3,T0,S3),(S3,T0,S4),
(S3,T0,S5),(S4,T0,S0),(S4,T0,S1),(S4,T0,S3),(S4,T0,S5)}
Minimising the values of sorts...
...New valuation found:
T -> {T0}
S -> {S0,S1,S2,S3}
QS -> {(S1,T0,S0),(S1,T0,S2),(S3,T0,S0),(S3,T0,S2)}
Minimising the values of sorts...
...New valuation found:
T -> {T0}
S -> {S0,S1,S2}
QS -> {(S1,T0,S0),(S1,T0,S1),(S1,T0,S2),(S2,T0,S0),(S2,T0,S1),(S2,T0,S2)}
Minimising the values of sorts...
...New valuation found:
T -> {T0}
S -> {S0,S1}
QS -> {(S0,T0,S0),(S0,T0,S1),(S1,T0,S0),(S1,T0,S1)}
Minimising the values of sorts...
...Done!
Minimising the values of predicates...
...New valuation found:
T -> {T0}
S -> {S0,S1}
QS -> {(S0,T0,S0),(S1,T0,S0)}
Minimising the values of sorts...
...Done!
The size of the cut-off set is now 5.

Computing the extended valuations...
...Done!
Found 4 non-isomorphic valuations.
(#valuations: 22, #canonical forms: 18, max #valuations stored: 7, #isomorphs removed: 0,
#branches pruned: 0)
Searching for satisfying valuations...
...New valuation found:
T -> {T0}
S -> {S0,S1,S2,S3,S4,S5,S6}
QS -> {(S3,T0,S1),(S3,T0,S2),(S3,T0,S4),(S3,T0,S6),(S5,T0,S0),(S5,T0,S1),(S5,T0,S4),(S5,T0,S6)}
Minimising the values of sorts...
...New valuation found:
T -> {T0}
S -> {S0,S1,S2,S3,S4}
QS -> {(S0,T0,S2),(S0,T0,S4),(S1,T0,S2),(S1,T0,S4),(S3,T0,S4)}
Minimising the values of sorts...
...New valuation found:
T -> {T0}
S -> {S0,S1,S2,S3}
QS -> {(S1,T0,S0),(S1,T0,S3),(S2,T0,S3)}
Minimising the values of sorts...
...New valuation found:
T -> {T0}
```

```
S -> {S0,S1,S2}
QS -> {(S1,T0,S0),(S1,T0,S1),(S2,T0,S0)}
Minimising the values of sorts...
...Done!
Minimising the values of predicates...
...New valuation found:
T -> {T0}
S -> {S0,S1,S2}
QS -> {(S0,T0,S2),(S1,T0,S2)}
Minimising the values of sorts...
...Done!
The size of the cut-off set is now 6.

Computing the extended valuations...
...Done!
Found 2 non-isomorphic valuations.
(#valuations: 38, #canonical forms: 26, max #valuations stored: 13, #isomorphs removed: 3,
#branches pruned: 0)
Searching for satisfying valuations...
...All found.
 The optimal cut-off set is found.

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

Generating Instance 2 generated by valuation
T -> {T0}
S -> {S0,S1}
QS -> {(S1,T0,S0)}

Generating Instance 3 generated by valuation
T -> {T0}
S -> {S0,S1}
QS -> {}

Generating Instance 4 generated by valuation
T -> {T0}
S -> {S0,S1,S2}
QS -> {}

Generating Instance 5 generated by valuation
T -> {T0}
S -> {S0,S1,S2}
QS -> {(S0,T0,S2),(S1,T0,S2)}

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

Instance 0 written successfully to file examples/raft_leader_election4_instance_0.csp

Instance 3 written successfully to file examples/raft_leader_election4_instance_1.csp
```

```
Instance 2 written successfully to file examples/raft_leader_election4_instance_2.csp

Instance 1 written successfully to file examples/raft_leader_election4_instance_3.csp

Instance 4 written successfully to file examples/raft_leader_election4_instance_4.csp

Instance 5 written successfully to file examples/raft_leader_election4_instance_5.csp

Checking examples/raft_leader_election4_instance_0.csp, this may take a while or two...

Check of examples/raft_leader_election4_instance_0.csp passed.

Checking examples/raft_leader_election4_instance_1.csp, this may take a while or two...

Check of examples/raft_leader_election4_instance_1.csp passed.

Checking examples/raft_leader_election4_instance_2.csp, this may take a while or two...

Check of examples/raft_leader_election4_instance_2.csp passed.

Checking examples/raft_leader_election4_instance_3.csp, this may take a while or two...

Check of examples/raft_leader_election4_instance_3.csp passed.

Checking examples/raft_leader_election4_instance_4.csp, this may take a while or two...

Check of examples/raft_leader_election4_instance_4.csp passed.

Checking examples/raft_leader_election4_instance_5.csp, this may take a while or two...

Check of examples/raft_leader_election4_instance_5.csp passed.

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

The total number of instances generated: 6
    (the total number of valuations generated: 186
     the total number of canonical forms computed: 127
     the maximum number of valuations stored all at once: 15)
Total time taken: 0.485503 seconds
    (time taken by input processing: 0 seconds
     time taken by the computation of valuations: 0.153 seconds
     time taken by output processing: 0.002 seconds
     time taken by trace refinement checking: 0.329 seconds)
```