Automating and Mechanising Cutoff Proofs for Parameterized Verification of Distributed Protocols

Shreesha G. Bhat
Indian Institute of Technology Madras, India

Kartik Nagar
Indian Institute of Technology Madras, India

Abstract

We propose a framework to automate and mechanize simulation-based proofs of cutoffs for parameterized verification of distributed protocols. We propose a strategy to derive the simulation relation given the cutoff instance and encode the correctness of the simulation relation as a formula in first-order logic. We have successfully applied our approach on a number of distributed protocols.

2012 ACM Subject Classification Software and its engineering → Software verification

Keywords and phrases Formal Methods, Automated Verification, Distributed Protocols

Acknowledgements We thank the organizers of the Young Research Fellow Program IITM and the 79ers for providing the required support and opportunity.

1 Introduction

The problem of parameterized verification [1] of distributed protocols asks whether a protocol satisfies its specification for all values of the parameter. Here, the parameter is typically the number of nodes involved in the protocol[1]. Cutoff based approaches for parameterized verification rely on the following observation: If the protocol can break its specification for some value of the parameter, it is guaranteed to break the specification for a value $\leq k$, where k is also called the cutoff. Small cutoffs can then enable fully automated verification, for example by exhaustively model checking all instances of the protocol of size $\leq k$.

In the recent past, there has been a lot of interest in automated and mechanised verification of distributed protocols [12, 4, 13, 7, 9, 11]. Most of these approaches rely on constructing and proving some form of inductive invariant, which should hold at each step of the protocol, and which is strong enough to imply the specification. This has resulted in highly useful frameworks [12, 9] for formally specifying the protocol description and the specification in the same language, which in turn has greatly facilitated fast and streamlined verification of complex protocols [11]. A number of approaches have also been proposed to automatically generate inductive invariants [7, 4].

In this work, our aim is to facilitate mechanized and automated verification of distributed protocols using a cutoff-based approach. While previous works have attempted to use cut-off based approaches for verification [3, 6, 8, 1], they have mostly been limited to either a restricted class of protocols or a restricted class of specifications. Recent examples of such works include [6] which is restricted to protocols which use consensus as a building block and [8] which only focuses on proving consensus-related specifications. All of these works enforce a specialized model of communication/execution. While it is known theoretically that all correct protocols have a finite cutoff [10], finding the cutoff is not an easy task. Our goal is to simplify this process by (1) providing a framework to mechanize the proof of correctness for

[1] There are also other possibilities such as number of values in the domains used by the protocol

a given cutoff and (2) by automating the verification process as much as possible. Further, we do not place any restrictions on the communication/execution model.

Towards this end, we show how to mechanise and automate simulation-based proofs for cutoffs using SMT solvers [2]. Such proofs involve constructing a simulation relation between the states of an arbitrary instance of the protocol and a smaller instance whose size is equal to the cutoff. The simulation must hold at every step of the protocol, and it must be sufficiently strong to prove the specification. Mechanising the proofs greatly increases the confidence in their correctness as compared with pen-and-paper proofs. We essentially reduce the cutoff problem to the satisfiability problem of first-order logic formulae, and if the protocol description, and the specification can be expressed in RML [12], then we ensure that the resulting formula will be in a decidable fragment of first-order logic. To the best of our knowledge, this is the first attempt to mechanize such proofs.

We also propose a principled approach to automatically synthesize the simulation relation. Our main observation is that the cutoff should be chosen in such a way that the actions leading to the violation of the specification can occur in the smaller instance of the protocol, independently of the actions in the larger instance. We derive the simulation relation directly from the protocol description, by using a direct correspondence between the nodes in the larger instance and the cutoff instance of the protocol. Finally, we observe that we only need to maintain the simulation relation up to the first violation of the specification, and hence, we can assume the specification to prove the simulation relation. In essence, we show that the simulation relation being an inductive invariant of the combined larger and smaller instances of the protocol relative to the specification is a sufficient condition for verification. We have applied the proposed approach successfully on a variety of distributed protocols including ring-based protocols, consensus protocols, protocols over an unreliable network, etc. and have proved low cutoffs (documented for the first time in many cases, to the best of our knowledge).

In this work, we make the following contributions:

- We develop a methodology for mechanising simulation-based proofs of cutoffs.
- We identify a principled strategy to determine the simulation relation based on a direct mapping between nodes of an arbitrary instance to the nodes of the cutoff instance.
- We identify a novel sufficient condition for simulation-based proofs of cutoffs, based on simulating up to the first violation of the specification.
- We have applied the proposed approach on a variety of distributed protocols.

2 Proposed Technique

Model We consider distributed protocols modelled in RML [12] where the system state is represented by a set of relations. The communication model between nodes is assumed to be asynchronous message passing. A set of actions are defined with guards and each step of the protocol involves non-deterministically firing one of these actions in an atomic fashion. The specification for the protocol is given as a safety property. The parameterized verification problem then asks whether for all instances of the protocol, does the specification hold at every step. We are interested in a cutoff on the number of nodes.

Inputs We assume that the protocol designer provides the proposed framework with, (1) The protocol description, (2) A cutoff instance, (3) A mapping from nodes of any arbitrary system to nodes of the cutoff instance given by sim, (4) Two functions Ω and τ as described below. Let C be the cutoff instance and $\mathbb{M}_C$ the set of nodes in C. Consider an arbitrary instance L of the protocol where $\mathbb{M}_L$ is the set of nodes in L such that $|\mathbb{M}_L| > |\mathbb{M}_C|$. The

mapping function sim has the following meaning, for each node $N_L \in \mathbb{M}_L$, N_L is simulated by $sim(N_L) \in \mathbb{M}_C$. The key intuition here is that a node $N_C \in \mathbb{M}_C$ effectively maintains the state components relevant to the violation of the safety property for all nodes $N_L \in \mathbb{M}_L$ such that $sim(N_L) = N_C$. To show that $|\mathbb{M}_C|$ is the cutoff, we will show that for some sequence of actions which leads to the first violation of the specification in any instance L of size $|\mathbb{M}_L| > |\mathbb{M}_C|$, there also exists a sequence of actions in the cutoff system of size $|\mathbb{M}_C|$ which leads to a violation.

Simulation Relation Our observation is that a generic form of the simulation relation can be given in-terms of the sim function. Let σ_L and σ_C denote the states of two instances L and C respectively. We can in general view σ_X as a function from nodes of the instance to state components. The simulation R between states maintains the property that all effects of actions that can contribute to a violation and are present in the state of a node in L must be present in the state of its simulating node C. This can be mathematically stated as follows:

$$(\sigma_L, \sigma_C) \in R \Leftrightarrow \forall N \in \mathbb{M}_L.\, \Omega(\sigma_L(N)) \subseteq \sigma_C(sim(N))$$

The relation uses the function Ω, which filters out those state components (i.e. effects of actions) which do not contribute in any way to the violation of the specification.

Lock Step The lock-step describes the action(s) taken in C for every action taken in L. The generic strategy behind the lock-step is that actions involving any two nodes L_1 and L_2 in L are translated to actions involving $sim(L_1)$ and $sim(L_2)$ in C. Note that this might result in some steps where $sim(L_1) = sim(L_2)$, which represents a *stuttering step* where L transitions according to the action but C stays in the same state. Stuttering steps can also occur when L performs some actions that C cannot perform. Similarly, a single action in L might need to be simulated by more than one action in C. This behaviour can be encapsulated in a function τ. In general, action a in L is translated to action $\tau(a)$ in C, where $\tau(a)$ can be a sequence of zero or more actions where zero actions represents a stuttering step.

FOL Encoding & Theorem To prove that the simulation relation holds at each step, we show that it is an inductive invariant of the combined instances L and C. Given FOL encoding of the states and actions of the protocol, we construct the following FOL formula to check the correctness of the simulation relation:

$$(\sigma_L, \sigma_C) \in R \wedge a(\sigma_L, \sigma'_L) \wedge \tau(a)(\sigma_C, \sigma'_C) \wedge (\sigma'_L, \sigma'_C) \notin R \tag{1}$$

Here, a can be any of the actions possible in L according to the protocol description, and we use the notation $a(\sigma_X, \sigma'_X)$ to denote the change in state after the action.

We construct a FOL encoding consisting of the protocol states, actions and the simulation relation along with the formula 1. If the resulting formula is `UNSAT`, then the simulation relation holds at every step. Note that we are only interested in the first violation of the specification in any arbitrary instance, because once a single violation occurs, the specification is broken and the protocol is incorrect, therefore, if Φ denotes the specification, we also conjunct $\Phi(\sigma_L)$ and $\Phi(\sigma_C)$ to the above formula. We also need to show that violations would be preserved by the simulation relation:

$$\neg\Phi(\sigma_L) \wedge R(\sigma_L, \sigma_C) \wedge \Phi(\sigma_C) \tag{2}$$

▶ **Theorem 1.** *If the formulae 1 and 2 are unsatisfiable, and if the cutoff instance C does not violate the specification Φ, then no instance of the protocol violates Φ.*

3 Demonstrative Example

Protocol Description. We refer to the Leader Election in a Ring protocol as in [12]. The system consists of a finite number of nodes in a ring setting, where each node can send a message to exactly 1 other node (also called its neighbour). Each node maintains a pending queue of messages, and we assume that messages are not lost. Each node has a unique ID with a total order on the IDs, and a node N with ID n and neighbour $NG(N)$ can perform one of two actions

1. $\texttt{generate}(N, ID(N), NG(N))$: Send a message $ID(N)$ to its neighbour $NG(N)$
2. $\texttt{handle\_message}(N, m, NG(N))$:
   - $m > ID(N)$: Forward the pending message m in its queue to its neighbour $NG(N)$
   - $m < ID(N)$: Remove the message m from its pending queue
   - $m = ID(N)$: Node N is elected as a leader (denoted by $\texttt{leader}(N)$)

Each step of the protocol corresponds to an action taken by one of the nodes. The state of the system consists of two components per node: the queue of pending messages at the node (denoted by the binary relation `pnd`), and the unary relation `leader`, which indicates whether the node has become a leader or not. The specification for the protocol is that there is at most one leader i.e. $\forall N_1, N_2. \neg(\texttt{leader}(N_1) \wedge \texttt{leader}(N_2) \wedge (N_1 \neq N_2))$.

Cutoff The cutoff of the protocol is actually 2, which means that it is sufficient to verify all 2-node instances of the protocol. We now demonstrate how we can determine the cutoff and prove its correctness. First, note that we are only really interested in the first violation of the specification in any arbitrary instance, because once a single violation occurs, the specification is broken and the protocol is incorrect. Since at every step of the protocol, at most one node can become a leader, the first violation of the specification would have exactly 2 nodes as leaders. Now, such a violation can also occur in an instance of size 2. To show a cutoff of 2, we will show that for some sequence of actions which leads to the first violation in any instance of size $n > 2$, there also exists a sequence of actions in a system of size 2 which leads to a violation. The property which enables this result is that all actions which contribute to the first violation in any arbitrary instance can be simulated in a 2-node instance[2] . Consider an arbitrary instance L of the protocol with more than 2 nodes. Let $\mathbb{M}_L$ be the set of nodes in L. Assume that a sequence of actions leads to a state where two nodes L_A and L_B with ID's A and B respectively are elected as leaders in L. We construct the cutoff system C with nodes $\mathbb{M}_S = \{C_A, C_B\}$ with ID's A and B.

Simulation Relation To define the *sim* relation, we use the ring topology, which is expressed in the form of a ternary relation `btw` such that $\texttt{btw}(X, Y, Z)$ holds whenever node Y lies between node X and Z in the ring, where the direction of message transfer in the ring is from X to Y. We use the `btw` relation to define the $sim : \mathbb{M}_L \to \mathbb{M}_S$ function as follows: For an arbitrary node $N \in \mathbb{M}_L$, if $\texttt{btw}(L_A, N, L_B) \vee N = L_B$ then $sim(N) = C_B$, else if $\texttt{btw}(L_B, N, L_A) \vee N = L_A$ then $sim(N) = C_A$. In essence, all the nodes between L_A and L_B in L including L_B are simulated by C_B and all the nodes between L_B and L_A including L_A are simulated by the node L_A.

[2] We note that this can be used as a generic strategy to obtain the cutoffs i.e. to look at possible violations of the specification and choose the smallest system that can exhibit all possible violations. In some cases as with the Sharded Key Value Store protocol [4], we find that introduction of some dummy nodes is required in the cutoff instance which simulate the state of multiple nodes in the larger system. We plan to expand upon these ideas to synthesize candidate cutoff instances as well.

For our example of Leader Election protocol, σ_X consists of the relations $\texttt{pnd}_X$ and $\texttt{leader}_X$ (for $X = L, C$). The generic definition described in 2 then translates into the following:

$$\begin{aligned}(\sigma_L, \sigma_C) \in R \iff \forall N \in \mathbb{M}_L.\, &(\texttt{leader}_L(L_A) \rightarrow \texttt{leader}_C(C_A)) \wedge (\texttt{leader}_L(L_B) \rightarrow \texttt{leader}_C(C_B)) \\ &\wedge (\neg\texttt{leader}_L(L_A) \wedge \texttt{pnd}_L(A, N) \rightarrow \texttt{pnd}_C(A, sim(N))) \\ &\wedge (\neg\texttt{leader}_L(L_B) \wedge \texttt{pnd}_L(B, N) \rightarrow \texttt{pnd}_C(B, sim(N)))\end{aligned}$$

The relevant state components which lead to a violation of the specification are only the messages containing IDs A and B. This is because only the receipt of these messages can cause either nodes L_A or L_B to become a leader, and the presence of other messages in the pending message queue of nodes do not affect their behavior on receiving messages A or B. Notice that in the cutoff instance C, only the messages containing IDs A and B can be passed, and only A or B can become leaders. Hence, Ω essentially filters exactly those state components from σ_L which can be present in σ_C.

Lock-Step According to our generic strategy, the lock-step can be encapsulated in a function τ which gives actions taken in the cutoff system C for every possible action in the arbitrary size system L. Our observation is that actions between nodes u_L and v_L are translated to actions between $sim(u_L)$ and $sim(v_L)$. Consistent with this observation, we obtain the following lock-step for Leader Election in a Ring.

$$\begin{aligned}\tau(\texttt{generate}_L(L_A, A, NG(L_A))) &= \texttt{generate}_C(C_A, A, C_B) \\ \tau(\texttt{handle\_message}_L(L_A, A, NG(L_A))) &= \texttt{handle\_message}_C(C_A, A, C_B) \\ \tau(\texttt{handle\_message}_L(L_A, B, NG(L_A))) &= \texttt{handle\_message}_L(C_A, B, C_B)\texttt{generate}_C(C_B, B, C_A) \\ \tau(\texttt{generate}_L(L_B, B, NG(L_B))) &= \texttt{generate}_C(C_B, B, C_A) \\ \tau(\texttt{handle\_message}_L(L_B, B, NG(L_B))) &= \texttt{handle\_message}_C(C_B, B, C_A) \\ \tau(\texttt{handle\_message}_L(L_B, A, NG(L_B))) &= \texttt{handle\_message}_L(C_B, A, C_A)\texttt{generate}_C(C_A, A, C_B)\end{aligned}$$

Note that in some of the cases above, a single action in L is simulated as multiple actions in C. This is required to ensure that the simulation relation is maintained.

Intuitive Reasoning behind the correctness of the Simulation Relation The simulation relation is maintained because: if we consider the message A, then as soon as it is generated by the node L_A in L and sent to its neighbour, the same message will be generated in the cutoff instance by the node C_A and sent to C_B. Now, A may be forwarded multiple times in portion of the ring between L_A and L_B, which would all correspond to stuttering steps of C, thus maintaining the simulation relation. When A finally reaches L_B in L and L_B handles the message, the lockstep actions of L and C would be the same, and would determine whether A is forwarded to the ring section between L_B and L_A in L, and to the pending queue of C_A in C. If L_B does forward the message A to its neighbour in L, the corresponding action at node C_B in C, apart from forwarding A to its neighbour C_A, also replenishes its pending queue itself with the message A. This is also done to ensure that the simulation relation continues to hold. If the message A were to eventually reach the node L_A in L resulting in its election as the leader, the same action occurs at C_A resulting in its election as a leader. A similar argument applies to the message B.

4 Experiments and Future Work

The tool implementing the ideas described in the paper is still a work-in-progress. However, we have some positive preliminary results: we have been able to verify the leader election

protocol and the significantly more complicated sharded key-value store protocol [4]. We use Z3 [2] as a back-end SMT solver, and in both cases, the verification time is in the order of a few seconds. In addition, we have manually checked the correctness of our approach on a variety of other protocols: Lock Service [14], Learning Switch [12], Distributed Lock Service [5]. As part of future work, we plan to finish the tool and apply our method on more complex protocols. In addition, we also want to leverage our observation regarding the relation between the cutoff instance and violations of the specification to automatically synthesize the cutoff instance.

To conclude, in this work, we have proposed an approach to significantly simplify and automate cut-off based proofs for verification of distributed protocols. Our experience is that cutoff-based proofs can be applied to a large number of distributed protocols, and we hope that this work would pave the way for more widespread application of this proof technique.

References

1 Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015.

2 Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

3 E. Allen Emerson and Kedar S. Namjoshi. Reasoning about rings. In *POPL*, pages 85–94. ACM Press, 1995.

4 Yotam M. Y. Feldman, James R. Wilcox, Sharon Shoham, and Mooly Sagiv. Inferring inductive invariants from phase structures. In *CAV (2)*, volume 11562 of *Lecture Notes in Computer Science*, pages 405–425. Springer, 2019.

5 Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jacob R. Lorch, Bryan Parno, Michael L. Roberts, Srinath T. V. Setty, and Brian Zill. Ironfleet: proving practical distributed systems correct. In *SOSP*, pages 1–17. ACM, 2015.

6 Nouraldin Jaber, Swen Jacobs, Christopher Wagner, Milind Kulkarni, and Roopsha Samanta. Parameterized verification of systems with global synchronization and guards. In *CAV (1)*, volume 12224 of *Lecture Notes in Computer Science*, pages 299–323. Springer, 2020.

7 Haojun Ma, Aman Goel, Jean-Baptiste Jeannin, Manos Kapritsos, Baris Kasikci, and Karem A. Sakallah. I4: incremental inference of inductive invariants for verification of distributed protocols. In *SOSP*, pages 370–384. ACM, 2019.

8 Ognjen Maric, Christoph Sprenger, and David A. Basin. Cutoff bounds for consensus algorithms. In *CAV (2)*, volume 10427 of *Lecture Notes in Computer Science*, pages 217–237. Springer, 2017.

9 Kenneth L. McMillan and Oded Padon. Ivy: A multi-modal verification tool for distributed algorithms. In *CAV (2)*, volume 12225 of *Lecture Notes in Computer Science*, pages 190–202. Springer, 2020.

10 Kedar S. Namjoshi. Symmetry and completeness in the analysis of parameterized systems. In *VMCAI*, volume 4349 of *Lecture Notes in Computer Science*, pages 299–313. Springer, 2007.

11 Oded Padon, Giuliano Losa, Mooly Sagiv, and Sharon Shoham. Paxos made EPR: decidable reasoning about distributed protocols. *Proc. ACM Program. Lang.*, 1(OOPSLA):108:1–108:31, 2017.

12 Oded Padon, Kenneth L. McMillan, Aurojit Panda, Mooly Sagiv, and Sharon Shoham. Ivy: safety verification by interactive generalization. In *PLDI*, pages 614–630. ACM, 2016.

13 Marcelo Taube, Giuliano Losa, Kenneth L. McMillan, Oded Padon, Mooly Sagiv, Sharon Shoham, James R. Wilcox, and Doug Woos. Modularity for decidability of deductive verification with applications to distributed systems. In *PLDI*, pages 662–677. ACM, 2018.

14 James R. Wilcox, Doug Woos, Pavel Panchekha, Zachary Tatlock, Xi Wang, Michael D. Ernst, and Thomas E. Anderson. Verdi: a framework for implementing and formally verifying distributed systems. In *PLDI*, pages 357–368. ACM, 2015.