paraVerifier: An Automatic Framework for Proving Parameterized Cache Coherence Protocols

Yongjian Li[1,3], Jun Pang[2], Yi Lv[1], Dongrui Fan[4],
Shen Cao[1], and Kaiqiang Duan[1]

[1] Institute of Software, Chinese Academy of Sciences, China
[2] Computer Science and Communications, University of Luxembourg, Luxembourg
[3] College of Information Engineering, Capital Normal University, Beijing, China
[4] Institute of Computing Technology, Chinese Academy of Sciences, China

Abstract. Parameterized verification of cache coherence protocols is an important but challenging research problem. We present in this paper our automatic framework paraVerifier to handle this problem: (1) it first discovers auxiliary invariants and the corresponding causal relations between invariants and protocol rules from a small reference instance of the verified protocol; (2) the discovered invariants and causal relations can then be generalized into their parameterized form to automatically construct a formal proof to establish the correctness of the protocol. paraVerifier has been successfully applied to a number of benchmarks.

1 Introduction

Verification of parameterized systems (e.g., see [1,2,3,4,5,6,7,8,9,10,11]) is interesting in the area of formal methods, mainly due to the practical importance of such systems. Parameterized systems exist in many application domains, including cache coherence protocols, security systems, and network communication protocols. In this work, we focus on cache coherence protocols, which play a key role in modern computer architecture. They require complex algorithms that deal with asynchrony, unpredictable message delays, and multiple communication paths between many nodes. Therefore, the highest possible assurance for the correctness of these systems should be guaranteed by formal reasoning techniques.

The challenge posed by parameterized verification of cache coherence protocols is that the desired safety properties, in terms of invariants, should hold for any instance of the studied protocol, not just for a single protocol instance. Model checking is automatic but is only able to verify an instance of the protocol. The correctness of the reference instance does not formally suffice to conclude the correctness for all instances. Due to the extreme importance of cache coherence protocols, it is preferable to have a proof for any instance of such protocols.

Advanced verification techniques such as compositional [12] and abstraction model checking [6] have been proposed to handle this challenge. However, auxiliary invariants of a cache coherence protocol, which is usually provided by a human, based on his insights of the protocol, are needed to make these techniques work. How to find sufficient and necessary invariants is the main difficulty in the field of parameterized verification.

Many works have focused on the construction of a set of auxiliary invariants, for example, see [4,6,8,10,11]. However, the theoretical foundation of these techniques and their soundness proofs are often only discussed in the respective papers. These theories themselves are not easy to understand, and are subjects to be mechanically checked, mainly due to the fact that their soundness needs to be guaranteed without any conditions.

The aim of our framework paraVerifier is to solve the parameterized verification of cache coherence protocols in a *unified*, *rigorous* and *automated* way. paraVerifier consists of two parts: an invariant finder invFinder and a proof generator proofGen. In order to verify that an invariant *inv* holds for any instance of a parameterized protocol, a reference model of the protocol with a fixed parameter is constructed first and successfully model checked, and invFinder tries to search for interesting auxiliary invariants and causal relations which are capable of proving *inv*. Next, proofGen explores the outputs of invFinder to construct a complete and parameterized formal proof in a theorem prover (e.g., Isabelle). Such a proof can eventually be checked automatically.

The originality of our work lies in the following aspects. First, paraVerifier is built on a simple but elegant theory. Three types of causal relations between protocol rules and invariants are identified, which are essentially the special cases of the general induction rule. The correctness of the three causal relations is captured by the so-called *consistency lemma*. It is heuristics-inspired by trying to construct the consistency relation that guides the tool invFinder to find auxiliary invariants. On the other hand, the consistency lemma provides a general guiding principle to prove invariants in the parameterized model of a cache coherence protocol. The lemma itself is verified as a formal theory in Isabelle [13].[5] Second, paraVerifier produces a list of invariants and a readable proof script for a given parameterized cache coherence protocol. The invariants are visible, in the sense that they can characterize the semantical features of the protocol and help users to precisely understand the design of the protocol. The formal proof script models the protocol rigorously and specifies its properties without any ambiguity, and more importantly it is mechanically checked. Third, paraVerifier is automatic, i.e., requiring little human intervention, and scalable. After the protocol is modeled in paraVerifier, auxiliary invariants are searched automatically via invFinder. The formal proof script in Isabelle is also automatically generated by proofGen, and checked by Isabelle. paraVerifier is successfully applied to industrial case studies such as the Flash protocol [14,12].[6]

2 Consistency Lemma

In this section, we introduce the theoretical foundation underlining paraVerifier. Consider a set of state variables V, we use e, f and S to denote an expression, a formula, and a statement over the set of state variables V. Variables are divided into two classes: array variables or non-array (global) variables. A state s of a protocol is an instantaneous snapshot of its behavior given by a mapping from all variables in V to natural numbers.

[5] We directly use parts of our Isabelles theories to introduce definitions and lemmas in the paper.

[6] Flash is considered as a standard and difficult benchmark for any proposed method for parameterized verification, as Chou et al. [6] state "if the method works on Flash, then there is a good chance that it will also work on many real-world cache coherence protocols".

We write expEval $e\ s$ (and formEval $f\ s$) to denote the evaluation of the expression e (and formula f) at the state s. With a parallel assignment $S = \{x_i := e_i | i > 0\}$, we define preCond $S\ f = f[x_i := e_i]$, which substitutes each occurrence of x_i by e_i.

Protocols. A cache coherence protocol is formalized as a pair $(ini, rules)$, where (1) *ini* is an initialization formula; and (2) *rules* is a set of transition rules. Each rule $r \in rules$ is defined as $g \rhd S$, where g is a predicate, and S is a parallel assignment to distinct variables v_i with expressions e_i. We write pre $r = g$, and act $r = S$ if $r = g \rhd S$.

We identify three kinds of causal relations that are essentially the special cases of the general induction rule. Consider a transition rule r, a formula f, and a formula set F, the three causal relations are defined as follows:

Definition 1. *We define the following relations*

1. invHoldForRule$_1$ $f\ r \equiv$ pre $r \longrightarrow$ preCond f (act r)*;*
2. invHoldForRule$_2$ $f\ r \equiv f =$ preCond f (act r)*;*
3. invHoldForRule$_3$ $f\ r\ F \equiv \exists f' \in F$ *s.t.* $(f' \wedge$ (pre r)$) \longrightarrow$ preCond f (act r)*;*
4. invHoldForRule $f\ r\ F$ *represents a disjunction of* invHoldForRule$_1$, invHoldForRule$_2$ *and* invHoldForRule$_3$.

The first relation (invHoldForRule$_1$ $f\ r$) means that after rule r is executed, f should hold. The second relation (invHoldForRule$_2$ $f\ r$) intuitively means that none of the state variables in f is changed and the execution of rule r does not affect the evaluation of f. The third relation (invHoldForRule$_3$ $f\ r\ F$) states that there exists another formula (invariant) $f' \in F$ such that the conjunction of the guard of r and f' implies that f holds after the election of rule r. Essentially, the causal relations capture whether and how the execution of a particular protocol rule changes the protocol state variables appearing in an invariant. More importantly, the relations can be considered as induction proof tactics designed for automatic proof generation (for example, used by proofGen).

Intuitively, the disjunction of the above relations (invHoldForRule $f\ r\ F$) defines a causal relation, which can guarantee that if each formula f in F holds before the execution of the rule r, then f also holds after the execution of the rule. Secondly, it can be considered as a special inductive proof rule, which can be applied to prove that each formula in F holds for each protocol rule.

Definition 2. *A consistency relation, i.e.,* consistent *invs ini rules, that holds between a protocol* $(ini, rules)$ *and a set of invariants* $invs = \{inv_1, \ldots, inv_n\}$*, is defined as:*

- *For any invariant* $inv \in invs$ *and state s, if ini is evaluated as true at state s (i.e.,* formEval *ini s = true), then inv is also evaluated as true at the state s.*
- *For any* $inv \in invs$ *and* $r \in rules$*,* invHoldForRule *inv r invs.*

Now we proceed with formally stating the *consistency lemma* below. Namely, if the consistency relation consistent *invs ini rules* holds, then for any $inv \in invs$ *inv* holds for any reachable state s of a protocol $(ini, rules)$.

Lemma 1. *For a protocol* $(ini, rules)$*, we use* reachableSet *ini rules to denote the set of reachable states of the protocol. Given a set of invariants invs, we have*

$$[\![\text{consistent } invs\ ini\ rules; s \in \text{reachableSet } ini\ rules]\!] \Longrightarrow \forall inv \in invs.\text{formEval } inv\ s$$

We have built a general theory `cache.thy` in Isabelle to model cache coherence protocols [15], and the consistency lemma is also formally proved.

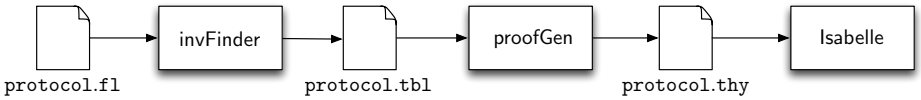

Fig. 1. The workflow of paraVerifier.

Table 1. An example fragment of `protocol.tbl` generated by invFinder.

protocol rule	ruleParas	invariant	causal relation	formula (f')
crit	[1]	inv1 1 2	invHoldForRule3	inv2 2
crit	[2]	inv1 1 2	invHoldForRule3	inv2 1
crit	[3]	inv1 1 2	invHoldForRule2	

3 Overview of Our Approach

The steps of our framework paraVerifier to parameterized verification of cache coherence protocol is illustrated in Fig. 1. A small cache coherence protocol instance `protoocl.fl`, is fed into the tool invFinder, which will search for all necessary ground auxiliary invariants from the reference protocol instance. A table `protoocl.tbl` is used to store the set of ground invariants and causal relations, which will then be used by proofGen to create an Isabelle proof script which models and verifies the protocol in a parameterized form. In this step, ground invariants will be generalized into a parameterized form, and accordingly ground causal relations will be adopted to create parameterized proof commands which essentially prove the existence of the parameterized causal relations. At last, the Isabelle proof script `protoocl.thy` is given to Isabelle to check the protocol correctness automatically.

The consistency lemma plays a crucial role in paraVerifier. It behaves as a heuristics to construct a consistency relation that guides the tool invFinder to find auxiliary invariants. On the other hand, it gives a general guiding principle to prove invariants of a cache coherence protocol. The consistency lemma eliminates the need of directly using the induction proof method. It allows us to focus on the causal relationship between transition rules of the protocol and its invariants. It also enables us to divide the proof of the invariants to a series of subproofs to verify whether one of the relations $\mathsf{invHoldForRule}_{1-3}$ hold for a rule and an invariant. The strategy of 'divide and conquer' is the key step to make the series of sub-proofs to be automated because the proof patterns for the subproofs are similar and modular. The tool proofGen will then automatically generate a proof that applies the the consistency lemma to prove correctness.

Starting from a given set of initial invariants, invFinder repeatedly tries to find new invariants, in the form of ground formulas, by constructing the causal relation between the invariants and the protocol rules. It uses an oracle[7] that checks whether a ground formula is an invariant in the small reference model of the protocol. invFinder stops until no new invariants can be found. The output of invFinder is stored in file `protocol.tbl`. Each line of the table records the index of an invariant, the name of a parameterized rule, the rule parameters to instantiate the rule, a causal relation between the invariant and a causal relation The table also records the proper formulas f' which is used to construct the third causal relation $\mathsf{invHoldForRule}_3$. An example of such table is shown in Tab. 1.

The formal Isabelle proof script `protoocl.thy` generated by proofGen includes the definitions of control signals, rules, invariants, initializing formula, lemmas and

[7] Implemented with SMV and the SMT solver Z3.

Table 2. Verification results on benchmarks.

Protocols	#rules	#invariants	time (seconds)	Memory (MB)
MESI	4	3	0.68	11.5
MOESI	5	3	0.65	23.2
Germanish [11]	6	3	0.68	23.0
German [6]	13	24	4.09	26.7
German with data [6]	15	50	12.05	29.4
Flash [14,12]	73	112	1457.42	169.4

their proofs. Here, we briefly explain the generalization principle involved in proofGen. For a ground invariant *inv*with parameters, proofGen analyzes the number of ground parameters in it and defines a parameterized invariant *pInv* by replacing the ground parameters with their corresponding symbolic parameters accordingly. Then proofGen explores symmetry relations and uses the following three relations ex1P or ex2p or ex3P to define all the actually parameterized invariants, where ex1P $N\ P \equiv \exists i.(i \leq N \wedge P\ i)$, ex2P $N\ P \equiv \exists i\ j.(i \leq N \wedge j \leq N \wedge i \neq j \wedge P\ i\ j)$, and ex3P $N\ P \equiv \exists i\ j\ k.(i \leq N \wedge j \leq N \wedge k \leq N \wedge i \neq j \wedge i \neq k \wedge j \neq k \wedge P\ i\ j\ k)$. For instance, for the formula $\neg(n[1] = C \wedge n[2] = C)$, two ground parameters 1 and 2 are extracted, and a formal invariant formula $inv_1\ i1\ i2 = \neg(n[i1] = C \wedge n[i2] = C)$ is defined by replacing 1 and 2 with symbolic parameters $i1$ and $i2$, and $\{f.\text{ex2P}\ N\ \lambda i1 i2.f = inv_1\ i1\ i2\}$ defines the set of all the formulas, each of which is symmetric to $inv_1\ 1\ 2$. The generalization of statements, rules, and causal relations can be defined accordingly. Each line in the ground causal relation table (Tab. 1), is generalized into a parameterized relation, which is the key to generate a proof command to select a proper causal relation to prove.

4 Validation and Conclusion

We implemented paraVerifier in Forte [16] and tested it on a number of cache coherence protocols. The detailed source codes and data can be found in [15]. Each experimental data includes the protocol model, the invariant sets, and the Isabelle proof script. Tab. 4 summarizes our verification results, recording the resources needed to compute the invariants and generate the proof scripts. Note that our proof of Flash is different from the one of Park et al. [14], where they need to manually construct an abstract transaction model of Flash. Our proof does not require this step and has less human interaction.

Within paraVerifier, our automatic framework for parameterized verification of cache coherence protocol, (1) instead of directly proving the invariants of a protocol by induction, we propose a general proof method based on the consistency lemma to decompose the proof goal into a number of small ones; (2) instead of proving the decomposed subgoals by hand, we automatically generate proofs for them based on the information computed in a small protocol instance.[8]

As we demonstrate in this work, combining theorem proving with automatic proof generation is promising in the field of formal verification of industrial protocols. Theorem proving can guarantee the rigorousness of the verification results, while automatic proof generation can release the burden of human interaction.

[8] Technical details of paraVerifier will be made available in a technical report.

Acknowledgments Yongjian Li, was supported by grants 61170073 and 61170304 and 2011DFG13000 from the National Natural Science Foundation of China.

References

1. Pnueli, A., Shahar, E.: A platform for combining deductive with algorithmic verification. In Porc. 16th Conference on Computer Aided Verification (CAV). LNCS 1102. Springer (1996) 184–195
2. Björner, N., Browne, A., Manna, Z.: Automatic generation of invariants and intermediate assertions. Theoretical Computer Science **173**(1) (1997) 49 – 87
3. Arons, T., Pnueli, A., Ruah, S., Xu, Y., Zuck, L.: Parameterized verification with automatically computed inductive assertions? In: Proc. 13th Conference on Computer Aided Verification (CAV). LNCS 2102. Springer (2001) 221–234
4. Pnueli, A., Ruah, S., Zuck, L.: Automatic deductive verification with invisible invariants. In: Proc. 7th Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). LNCS 2031. Springer (2001) 82–97
5. Tiwari, A., Rueß, H., Saïdi, H., Shankar, N.: A technique for invariant generation. In: Proc. 7th Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). LNCS 2031. Springer (2001) 113–127
6. Chou, C.T., Mannava, P., Park, S.: A simple method for parameterized verification of cache coherence protocols. In: Proc. 5th Conference on Formal Methods in Computer-Aided Design (FMCAD). LNCS 3312. Springer (2004) 382–398
7. Pang, J., Fokkink, W., Hofman, R., Veldema, R.: Model checking a cache coherence protocol of a Java DSM implementation. Journal of Logic and Algebraic Programming **71**(1) (2007) 1 – 43
8. Pandav, S., Slind, K., Gopalakrishnan, G.: Counterexample guided invariant discovery for parameterized cache coherence verification. In: Proc. 13th IFIP Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME). LNCS 3725. Springer (2005) 317–331
9. Lv, Y., Lin, H., Pan, H.: Computing invariants for parameter abstraction. In: Proc. 5th IEEE/ACM Conference on Formal Methods and Models for Codesign (MEMOCODE). IEEE CS (2007) 29–38
10. Bingham B.: Automatic non-interference lemmas for parameterized model checking. In Proc. 8th Conference on Formal Methods in Computer-Aided Design (FMCAD). IEEE CS (2008) 1–8
11. Conchon, S., Goel, A., Krstic, S., Mebsout, A., Zaïdi, F.: Cubicle: A parallel SMT-based model checker for parameterized systems. In Proc. 24th Conference on Computer Aided Verification (CAV). LNCS 7385. Springer (2012) 718–724
12. McMillan, K.L., Labs, C.B.: Parameterized verification of the Flash cache coherence protocol by compositional model checking. In: Proc. 9th IFIP Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME). LNCS 2144. Springer (2001) 179–195
13. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic. Volume 2283 of LNCS. Springer (2002)
14. Park, S., Dill, D.L.: Verification of Flash cache coherence protocol by aggregation of distributed transactions. In: Proc. 8th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA). ACM (1996) 288–296
15. Li, Y.: invFinder: An invariant finder (2014) `http://lcs.ios.ac.cn/~lyj238/invFinder.html`.
16. Technical Publications and Training, Intel Corporation: Forte/FL User Guide. 2003.