Invariant Synthesis: Decidable Fragments to the Rescue

Stephan Merz

University of Lorraine, CNRS, Inria, LORIA, Nancy, France

1 Background

Inductive invariants are a staple of the deductive verification of programs and algorithms: for example, verifiers such as Dafny [9], Viper [11] or Why3 [4] rely on programmers annotating loops and classes with invariants. When proving the correctness of concurrent and distributed algorithms, global inductive invariants characterize the set of reachable system states, and they are notoriously hard to come up with. For example, when verifying the Pastry algorithm [14] that implements a distributed hash table in a peer-to-peer network, more than 80 invariants were developed as part of an interactive proof of roughly 30,000 lines [1]. Beyond their fundamental role in correctness proofs for algorithms and systems, invariants provide insights into possible variants and improvements. Thus, the invariants developed for the Pastry proof suggested a simplification of the protocol for incoming nodes to join the overlay network.

Because finding inductive invariants is difficult, techniques for assisting proof engineers in designing them are of great interest. Modern symbolic model checking algorithms such as IC3/PDR [2, 3] can compute inductive invariants for finite-state systems as a byproduct of verification. For the verification of parameterized systems, one can use these techniques for computing invariants for finite instances and then try to generalize them. The IC3PO algorithm by Goel and Sakallah [6] discovers symmetries in the invariants computed for the finite-state case and introduces quantifiers for generalizing them to the parameterized specification. For example, suppose that when verifying an instance of a system with 4 nodes numbered $1\,..\,4$, IC3 generated an invariant of the form

$$\begin{array}{l} \wedge\, P(2) \Rightarrow Q(1) \\ \wedge\, P(3) \Rightarrow Q(1) \wedge Q(2) \\ \wedge\, P(4) \Rightarrow Q(1) \wedge Q(2) \wedge Q(3) \end{array}$$

then IC3PO can suggest the generalized invariant

$$\forall m, n \in 1\,..\,N : P(m) \wedge n < m \Rightarrow Q(n)$$

for a generic instance containing N nodes. IC3PO's input language is based on that of Ivy [10], and it relies on the Ivy model checker for inferring inductive invariants for finite instances. Computing symmetries in a set of clauses is efficient, and IC3PO handles fully symmetric domains, as well as linear orders, as in the above example. It can also introduce quantifier alternations. However, in general there is no formal guarantee that the inferred formula is indeed an invariant of the parameterized system. IC3PO employs a saturation loop by incrementing the parameter in order to weed out spurious invariant candidates, and to detect when the algorithm has stabilized.

2 Invariant Synthesis in Practice

In joint work with Goel and Sakallah [5], we used IC3PO for inferring an inductive invariant for the well-known Bakery algorithm. The inferred invariant turned out to be remarkably similar to a handwritten invariant used in an existing correctness proof of that algorithm [8]; it is actually a little more permissive than the original invariant.

A more ambitious project is the verification of the Raft consensus protocol [12], designed to tolerate crash faults. In Raft, every node stores a *log*, i.e. a sequence of entries (v, t) where v is a value and t is a term. Terms (natural numbers) designate periods for which a node was elected leader. A leader for term t appends new entries to its log and tries to disseminate its entries, say the entry at position i in the log, to other nodes. A non-leader node accepts an entry (v, t) at position i from a node whose term is t' if t' is at least high as its own term number (which it will then update to t' if necessary) and if $i = 0$ or if its entry at position $i - 1$ agrees with the entry at position $i - 1$ of the node pushing the new entry. Observe that this may lead to a previously existing entry at position i to be overwritten. When the leader for term t has pushed an entry (v, t) to a quorum (a majority of nodes), the entry is *committed*, as well as all entries that appear earlier in the log. A node that suspects the current leader to have failed, chooses a term number t' that is higher than its current term and becomes candidate for becoming the leader for term t', sending out vote requests to other nodes. A node votes for a candidate at term t' if t' is higher than its current term number, if it has not yet voted for any node at t', and if the log of the candidate is at least as up to date as its own log. A candidate is elected when it has received votes from a quorum of nodes. The following properties are to be verified:

Leader uniqueness. No two distinct nodes $n \neq n'$ can be leaders for any given term.
Agreement. If two nodes n, n' hold entries (v, t) and (v', t) at the i-th position of their log for the same term t, then $v = v'$.
Commitment. Any committed entry is held by a majority of nodes, including the current leader, and is therefore permanent.

Proving the uniqueness of leaders for any given term t is quite easy, given that any node submits at most one vote for term t, and that any two quorums intersect. Indeed, IC3PO quickly computes an inductive invariant that implies that property.

Proving the agreement property is harder. The first difficulty is how to represent the logs of nodes: note that the rule for accepting an entry refers to the preceding entry in the log, but this cannot directly be expressed in the theory of total orders supported by Ivy and IC3PO. Given uninterpreted sorts `node`, `value` and totally ordered sorts `term` and `index`, we can represent the logs of Raft nodes as two functions

$$\begin{aligned} &\mathit{log\_value} : \texttt{node} \times \texttt{index} \to \texttt{value} \\ &\mathit{log\_term} : \texttt{node} \times \texttt{index} \to \texttt{term} \end{aligned}$$

satisfying the predicate

$$(\mathit{log\_value}(N, I) = \mathit{nullv}) \Leftrightarrow (\mathit{log\_term}(N, I) = \mathit{nullt})$$

for constants nullv and nullt representing the absence of a value or term of an entry. In the initial state, no values and terms are present. In this representation, the logs may

contain "holes" in the sense that their entries at some positions i and j may be defined (different from null), yet all entries at positions k with $i < k < j$ may be undefined. In this representation, the position I of the predecessor of an entry at position J of node N can be identified using the relation

$$\begin{aligned} pred(N, I, J) \triangleq\ & \wedge I < J \wedge log\_value(N, I) \neq nullv \\ & \wedge \forall K.\, I < K \wedge K < J \Rightarrow log\_value(N, K) = nullv \end{aligned}$$

The second difficulty comes from the fact that entries may be overwritten, but in order to express a suitable inductive invariant, it is convenient to refer to old entries. We therefore represent not only the actual logs of nodes, but also "ghost entries" (v, t) that existed at some point during the execution but may have been overwritten. With these preparations, and relying on the inductive invariant established for leader election, IC3PO computes an inductive invariant that asserts the following facts:

- A log entry at a higher index position cannot have a lower term value than that at any lower position.
- The actual log is contained in the log of ghost entries.
- For any node n and ghost entry (v, t) at index position i, the following hold:
  - t is at most the current term of node n, and is strictly lower than the current term if n is currently a candidate for leader election,
  - there exists a server l that was elected by a quorum for term t such that (v, t) appears among the ghost entries of l at position i; moreover, if l is currently leader for term t, then the entry appears in the actual log of l at position i.
- If the logs of two nodes contain entries (v, t) at (v', t) at some index i, then the prefixes of their logs up to position i agree.

Observe that some quantified subformulas appear in this invariant, such as a server having obtained a quorum. In order for IC3PO to be able to generate the invariant, these subformulas had to be introduced as auxiliary definitions in the specification. The invariant underlying the commitment property is even more complex and relies on additional ghost variables. Its mechanical synthesis is the subject of ongoing work.

3 Concluding Remarks

Techniques for synthesizing inductive invariants are becoming applicable to non-trivial algorithms. They rely on a combination of symbolic verification techniques and heuristics, precise reasoning techniques, and even machine learning [13, 15, 17, 18].

Choosing decidable logical fragments underlying the synthesis techniques helps them to be robust. For example, Ivy is based on a generalization of the EPR fragment of first-order logic. It also limits the scope in which the techniques are applicable, although proof engineers may be able to push the boundaries, as in the "sequence with holes" abstraction used for the proof of Raft. In particular, a specification written in a decidable fragment is amenable to independent verification of the synthesized invariant candidate, and techniques such as IC3 have been extended directly beyond propositional logic.

In this perspective, the design of expressive and decidable fragments of first-order theories, as in [7, 16], is an important and practically relevant subject of scientific investigation, beyond pure theoretical curiosity. Happy birthday, Christoph!

References

1. N. Azmy, S. Merz, and C. Weidenbach. A machine-checked correctness proof for Pastry. *Sci. Comput. Program.*, 158:64–80, 2018.
2. A. R. Bradley. SAT-based model checking without unrolling. In R. Jhala and D. A. Schmidt, editors, *12th Intl. Conf. Verification, Model Checking, and Abstract Interpretation (VMCAI 2011)*, volume 6538 of *LNCS*, pages 70–87, Austin, TX, 2011. Springer.
3. N. Eén, A. Mishchenko, and R. K. Brayton. Efficient implementation of property directed reachability. In P. Bjesse and A. Slobodová, editors, *Intl. Conf. Formal Methods in Computer-Aided Design (FMCAD'11)*, pages 125–134, Austin, TX, 2011. FMCAD Inc.
4. J. Filliâtre and A. Paskevich. Why3 – where programs meet provers. In M. Felleisen and P. Gardner, editors, *22nd Europ. Symp. Programming Languages and Systems (ESOP 2013)*, volume 7792 of *LNCS*, pages 125–128, Rome, Italy, 2013. Springer.
5. A. Goel, S. Merz, and K. A. Sakallah. Towards an automatic proof of the bakery algorithm. In M. Huisman and A. Ravara, editors, *Formal Techniques for Distributed Objects, Components, and Systems (FORTE 2023)*, volume 13910 of *LNCS*, pages 21–28, Lisbon, Portugal, 2023. Springer.
6. A. Goel and K. A. Sakallah. Regularity and quantification: a new approach to verify distributed protocols. *Innov. Syst. Softw. Eng.*, 19(4):359–377, 2023.
7. M. Horbach, M. Voigt, and C. Weidenbach. On the Combination of the Bernays-Schönfinkel-Ramsey Fragment with Simple Linear Integer Arithmetic. In L. de Moura, editor, *26th Intl. Conf. Automated Deduction*, volume 10395 of *LNCS*, pages 77–94, Gothenburg, Sweden, 2017. Springer.
8. L. Lamport. A machine-checked proof of the Bakery algorithm. `https://github.com/tlaplus/Examples/blob/master/specifications/Bakery-Boulangerie/Bakery.tla`.
9. K. R. M. Leino. Dafny: An automatic program verifier for functional correctness. In E. M. Clarke and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-16)*, volume 6355 of *LNCS*, pages 348–370, Dakar, Senegal, 2010. Springer.
10. K. L. McMillan and O. Padon. Ivy: A multi-modal verification tool for distributed algorithms. In S. K. Lahiri and C. Wang, editors, *32nd Intl. Conf. Computer Aided Verification (CAV 2020)*, volume 12225 of *LNCS*, pages 190–202, Los Angeles, CA, 2020. Springer.
11. P. Müller, M. Schwerhoff, and A. J. Summers. Viper: A verification infrastructure for permission-based reasoning. In A. Pretschner, D. Peled, and T. Hutzelmann, editors, *Dependable Software Systems Engineering*, volume 50 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 104–125. IOS Press, 2017.
12. D. Ongaro and J. K. Ousterhout. In search of an understandable consensus algorithm. In G. Gibson and N. Zeldovich, editors, *Proc. USENIX Annual Technical Conference*, pages 305–319, Philadelphia, PA, 2014. USENIX Association.
13. G. Redondi, A. Cimatti, A. Griggio, and K. L. McMillan. Invariant checking for smt-based systems with quantifiers. *ACM Trans. Comput. Log.*, 25(4):1–37, 2024.
14. A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In R. Guerraoui, editor, *IFIP/ACM Intl. Conf. Distributed Systems Platforms (Middleware 2001)*, volume 2218 of *LNCS*, pages 329–350, Heidelberg, Germany, 2001. Springer.
15. W. Schultz, I. Dardik, and S. Tripakis. Plain and simple inductive invariant inference for distributed protocols in tla$^+$. In A. Griggio and N. Rungta, editors, *22nd Intl. Conf. Formal Methods in Computer-Aided Design (FMCAD 2022)*, pages 273–283, Trento, Italy, 2022. IEEE.

16. T. Sturm, M. Voigt, and C. Weidenbach. Deciding First-Order Satisfiability when Universal and Existential Variables are Separated. In *LICS 2016*, pages 86–95, New York, NY, 2016.
17. J. R. Wilcox, Y. M. Y. Feldman, O. Padon, and S. Shoham. mypyvy: A research platform for verification of transition systems in first-order logic. In A. Gurfinkel and V. Ganesh, editors, *36th Intl. Conf. Computer Aided Verification (CAV 2024)*, volume 14682 of *LNCS*, pages 71–85, Montreal, Canada, 2024. Springer.
18. T. N. Zhang, K. Singh, T. Chajed, M. Kapritsos, and B. Parno. Basilisk: Using provenance invariants to automate proofs of undecidable protocols. In L. Zhou and Y. Zhou, editors, *19th USENIX Symp. Operating Systems Design and Implementation (OSDI 2025)*, pages 1–17, Boston, MA, USA, 2025. USENIX Association.