VeriMon: A Formally Verified Monitoring Tool

David Basin[1], Thibault Dardinier[1], Nico Hauser[1], Lukas Heimes[1], Jonathan Julián Huerta y Munive[2], Nicolas Kaletsch[1], Srđan Krstić[1], Emanuele Marsicano[1], Martin Raszyk[3], Joshua Schneider[1], Dawit Legesse Tirore[4], Dmitriy Traytel[2(✉)], and Sheila Zingg[1]

[1] Department of Computer Science, ETH Zürich, Zurich, Switzerland
[2] Department of Computer Science, University of Copenhagen, Copenhagen, Denmark
traytel@di.ku.dk
[3] DFINITY Foundation, Zurich, Switzerland
[4] Computer Science Department, IT University of Copenhagen, Copenhagen, Denmark

Abstract. A runtime monitor observes a running system and checks whether the sequence of events the system generates satisfies a given specification. We describe the evolution of VeriMon: an expressive and efficient monitor that has been formally verified using the Isabelle proof assistant.

1 Introduction

The goal of runtime verification (RV) is to gain confidence in the correctness of a given execution of a running system. This is a lightweight alternative to full formal verification which must consider all possible executions. In RV, monitors are tools that take as input an execution represented as a sequence of events called trace, analyze the trace with respect to a given specification, and output verdicts, i.e., satisfactions or violations of the specification. Monitors support a wide range of specification languages [8], including automaton-based, (temporal-)logic-based, and (recursive-)rule-based formalisms.

A monitor's specification language must be *expressive* to allow users to formulate the desired properties in a concise, natural, and intuitive way. At the same time and often in direct conflict with the expressiveness requirement, monitors must be *time-* and *memory-efficient*. Expressive and efficient monitors use complex, optimized algorithms, whose correctness is not obvious. Yet a monitor must be *trustworthy* to be used as a verification tool.

VeriMon [1,18,20] is an expressive, efficient, and trustworthy monitor. Its specification language is based on the expressive metric first-order temporal logic (MFOTL) [2], but it additionally incorporates automata-based and rule-based features. It uses efficient algorithms for evaluating the temporal operators Since and Until, n-ary conjunctions (as multi-way joins), and aggregations such as

H. Seidl et al. (Eds.): ICTAC 2022, LNCS 13572, pp. 1–6, 2022.
https://doi.org/10.1007/978-3-031-17715-6_1

sums or averages. Finally, it is trustworthy as it has been formally verified using the Isabelle/HOL proof assistant [15]. Proof assistants are tools that mechanically check the correctness of human-written mathematical proofs, e.g., of an algorithm's correctness. They are built around a small, well-understood inference kernel through which all reasoning must pass, which provides the highest level of trustworthiness.

Here, we describe VeriMon's origins and evolution, outline some planned next steps, and discuss the advantages of formally verifying monitors.

2 Evolution

VeriMon originated from a certain dissatisfaction with existing monitoring tools. Specifically, we have been using the efficient MFOTL monitor MonPoly [2,3] for years. But every so often, we would discover and fix an implementation bug. While annoying, this was not the most pressing issue. More importantly, MonPoly became an impenetrable black box: extending its specification language or improving its algorithms became extremely difficult for us as the implementation included various undocumented and non-obvious performance optimizations and the original implementors had moved on. (A typical fate of academic software!)

Eventually, we decided to start from scratch aiming at establishing the correctness of a much simplified algorithm, which did not include performance optimizations and supported a restricted specification language. To this end, we formulated in Isabelle the syntax and semantics of MFOTL, defined a core monitoring algorithm as a functional program, and proved the algorithm sound (all produced verdicts are correct according to the semantics) and complete (all verdicts that hold under the semantics are eventually produced). To obtain an executable program, we used Isabelle's code generator [10] to extract 2 800 lines of OCaml code from our formalization consisting of 3 000 lines of Isabelle definitions and proofs. The extracted code included two main functions (and their dependencies): *init* that initialized the monitor's state for a given abstract syntax tree of an MFOTL specification and *step* that updates the monitor's state upon incoming events while outputting verdicts. The first version of VeriMon [18] augmented this verified core with MonPoly's unverified specification and log parsers and modules for type-checking, rewriting, and preprocessing specifications and for printing verdicts.

After this kick-start, the first target was to align VeriMon's variant of MFOTL with MonPoly's, which included inequalities and aggregation operators. Having formally established and thus understood the algorithmic invariants for other non-temporal operators made these extensions straightforward [1, §2–3]. At that point, we were in the position of extending VeriMon, one feature at a time, often carried out mostly by undergraduate students. Today, VeriMon incorporates:

- Regular expression matching operators generalizing MFOTL's temporal operators and representing a form of automaton-based specifications [1, §4];
- A non-recursive let operator, invaluable for structuring policies [20, §3];

– A recursive let operator that requires all recursive occurrences to be guarded by past temporal operators and can encode rule-based specifications [20, §4].

We are currently working on adding support for dual temporal operators (Release and Trigger) [13]. All these extensions do not only introduce new operators, but also extend the correctness proof to cover the new features.

The first version of VeriMon was extremely inefficient. We have spent considerable time and energy on verifying performance optimizations. VeriMon became the incubator for developing and proving correct algorithms for the evaluation of Since and Until [1, §6][16, §4.4] and aggregations over those that asymptotically improved over MonPoly's algorithms. We also used insights from databases and incorporated a worst-case optimal multi-way join algorithm [1, §5]. Overall, VeriMon still tends to be slower and use more memory than MonPoly, but it is easy to construct examples in which the better algorithms reverse the picture. In the meantime, some of these algorithms have also found their way into MonPoly.

We have also made progress on reducing the amount of VeriMon's unverified code by verifying a type inference algorithm and a specification rewriting module.

Since the first version, VeriMon's publicly available code base[1] grew significantly. The formalization now spans over 45 000 lines of Isabelle definitions and proofs. The extracted code amounts to over 11 000 lines of OCaml. Thanks to the transpiler `js_of_ocaml` [19], we can now run VeriMon in every web browser.[2] This is not recommended for realistic applications (due to the suboptimal performance and the increased trusted code base which then includes `js_of_ocaml` and the browser's JavaScript engine), but extremely useful for demonstrations.

3 Future Directions

We plan to improve VeriMon along the three discussed dimensions. For trustworthiness, the missing ingredients are the specification and log parsers. Once verified, they will allow us to run VeriMon without relying on MonPoly's unverified code.

To further improve efficiency, we will use database-style indices to speed up joins and other operations on tables, the main computations in VeriMon aside from the temporal operator evaluation. Furthermore, we plan to lift the recently developed algorithms for the regular expression matching operators in the propositional metric temporal logic setting [16, §3] to VeriMon's first-order setting.

In terms of expressiveness, we aim to generalize VeriMon's time domain from natural numbers to an arbitrary domain meeting minimal well-formedness conditions. This will improve the flexibility of VeriMon's metric intervals used to express quantitative temporal constraints. We also intend to verify and incorporate algorithms for a Datalog-style recursive let operator. Finally, VeriMon, like MonPoly, operates on finite tables and thus can only handle the *monitorable*

[1] https://bitbucket.org/jshs/monpoly/src/master/.
[2] https://traytel.bitbucket.io/verimon.

fragment of MFOTL [2, §4.2]. While other approaches, which can handle full MFOTL by replacing tables with automatic structures or binary decision diagrams, exist [2,11,12], we believe that working with finite tables is a major source of efficiency for VeriMon. Thus, we plan to verify and integrate in VeriMon the recent approach of rewriting arbitrary MFOTL specifications into the monitorable fragment [16, §4.3].

4 Discussion

The obvious benefit of working with a formally verified algorithm is the absence of bugs. This benefit is no longer given when the verified code is combined with unverified code. Indeed, we found and fixed several issues in the unverified glue code connecting VeriMon's data structures to MonPoly's in VeriMon's early days. Yet, the glue code is only a few hundred lines and it is much easier to localize the problem there compared to the thousands of lines of code comprising the actual monitor.

A bigger danger for verified tools are misunderstandings in the semantics. For example, VeriMon used to compute averages as $a + b/2$ because it reused the same faulty Isabelle definition in the semantics and the algorithm, which omitted a pair of parentheses by mistake. To avoid such issues, the formalized semantics of the specification language must be carefully inspected, including all auxiliary definitions. Fortunately, and again in contrast to the actual monitoring algorithm, VeriMon's semantics comprises only a few hundred lines of Isabelle definitions.

A major asset for VeriMon's usability is its tight integration with MonPoly. Both tools are compiled into a single binary, which distinguishes the used algorithm via a flag. This resulted in a standard workflow, in which users run MonPoly and rerun using VeriMon in case MonPoly's output looks suspicious. We have also performed such a comparative execution on a larger scale on random inputs. This differential testing revealed discrepancies [1,18,20] pointing to bugs and an unusual (but specified) semantics in the unverified tools MonPoly and DejaVu [12].

We see extensibility as the main advantage of a formally verified monitor. The verification of the first version of VeriMon has already identified several notions and their properties central to the verification. Adding new features then reduced to extending these notions while updating the proofs of their properties. Along similar lines, we replaced inefficient algorithms by efficient ones using refinement, which allowed us to reuse the proofs of the inefficient algorithms' correctness.

Several of VeriMon's features, such as the non-recursive let operator and the improved algorithms for Since and Until, have been propagated back to MonPoly and have guided the design of a new monitoring tool implemented in C++, CPPMon.[3]

We are happy to start seeing other work in the community that uses proof assistants [4–6,17], deductive verifiers [9], or SMT solvers [7,14] to improve the

[3] https://github.com/matthieugras/cppmon.

trustworthiness of monitors. We believe that formal verification is the only way towards a landscape of tools that are reliable and maintainable: not just one-paper wonders.

Acknowledgments. Research on VeriMon has been supported by the Swiss National Science Foundation grant "Big Data Monitoring" (167162), the US Air Force grant "Monitoring at Any Cost" (FA9550-17-1-0306), and a Novo Nordisk Foundation Start Package grant (NNF20OC0063462). The authors are listed in alphabetical order regardless of individual contributions or seniority.

References

1. Basin, D., Dardinier, T., Heimes, L., Krstić, S., Raszyk, M., Schneider, J., Traytel, D.: A formally verified, optimized monitor for metric first-order dynamic logic. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) IJCAR 2020. LNCS (LNAI), vol. 12166, pp. 432–453. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51074-9_25
2. Basin, D., Klaedtke, F., Müller, S., Zălinescu, E.: Monitoring metric first-order temporal properties. J. ACM **62**(2), 15:1–15:45 (2015). https://doi.org/10.1145/2699444
3. Basin, D., Klaedtke, F., Zălinescu, E.: The MonPoly monitoring tool. In: Reger, G., Havelund, K. (eds.) RV-CuBES 2017. Kalpa Publications in Computing, vol. 3, pp. 19–28. EasyChair (2017). https://doi.org/10.29007/89hs
4. Blech, J.O., Falcone, Y., Becker, K.: Towards certified runtime verification. In: Aoki, T., Taguchi, K. (eds.) ICFEM 2012. LNCS, vol. 7635, pp. 494–509. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34281-3_34
5. Bohrer, R., Tan, Y.K., Mitsch, S., Myreen, M.O., Platzer, A.: VeriPhy: verified controller executables from verified cyber-physical system models. In: Foster, J.S., Grossman, D. (eds.) PLDI 2018, pp. 617–630. ACM (2018). https://doi.org/10.1145/3192366.3192406
6. Chattopadhyay, A., Mamouras, K.: A verified online monitor for metric temporal logic with quantitative semantics. In: Deshmukh, J., Ničković, D. (eds.) RV 2020. LNCS, vol. 12399, pp. 383–403. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-60508-7_21
7. Dauer, J.C., Finkbeiner, B., Schirmer, S.: Monitoring with verified guarantees. In: Feng, L., Fisman, D. (eds.) RV 2021. LNCS, vol. 12974, pp. 62–80. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-88494-9_4
8. Falcone, Y., Krstić, S., Reger, G., Traytel, D.: A taxonomy for classifying runtime verification tools. Int. J. Softw. Tools Technol. Transf. **23**(2), 255–284 (2021). https://doi.org/10.1007/s10009-021-00609-z
9. Finkbeiner, B., Oswald, S., Passing, N., Schwenger, M.: Verified Rust monitors for Lola specifications. In: Deshmukh, J., Ničković, D. (eds.) RV 2020. LNCS, vol. 12399, pp. 431–450. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-60508-7_24
10. Haftmann, F., Nipkow, T.: Code generation via higher-order rewrite systems. In: Blume, M., Kobayashi, N., Vidal, G. (eds.) FLOPS 2010. LNCS, vol. 6009, pp. 103–117. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12251-4_9

11. Havelund, K., Peled, D., Ulus, D.: First order temporal logic monitoring with BDDs. In: Stewart, D., Weissenbacher, G. (eds.) FMCAD 2017, pp. 116–123. IEEE (2017). https://doi.org/10.23919/FMCAD.2017.8102249
12. Havelund, K., Peled, D., Ulus, D.: DejaVu: a monitoring tool for first-order temporal logic. In: MT@CPSWeek 2018, pp. 12–13. IEEE (2018). https://doi.org/10.1109/MT-CPS.2018.00013
13. Huerta y Munive, J.J.: Relaxing safety for metric first-order temporal logic via dynamic free variables. In: Thao, D., Stolz, V. (eds.) RV 2022. LNCS, Springer (2022) (to appear)
14. Laurent, J., Goodloe, A., Pike, L.: Assuring the guardians. In: Bartocci, E., Majumdar, R. (eds.) RV 2015. LNCS, vol. 9333, pp. 87–101. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23820-3_6
15. Nipkow, T., Wenzel, M., Paulson, L.C. (eds.): Isabelle/HOL - A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45949-9
16. Raszyk, M.: Efficient, Expressive, and Verified Temporal Query Evaluation. Ph.D. thesis, ETH Zürich (2022). https://doi.org/10.3929/ethz-b-000553221
17. Rizaldi, A., et al.: Formalising and monitoring traffic rules for autonomous vehicles in isabelle/HOL. In: Polikarpova, N., Schneider, S. (eds.) IFM 2017. LNCS, vol. 10510, pp. 50–66. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66845-1_4
18. Schneider, J., Basin, D., Krstić, S., Traytel, D.: A formally verified monitor for metric first-order temporal logic. In: Finkbeiner, B., Mariani, L. (eds.) RV 2019. LNCS, vol. 11757, pp. 310–328. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32079-9_18
19. Vouillon, J., Balat, V.: From bytecode to JavaScript: the js_of_ocaml compiler. Softw. Pract. Exp. **44**(8), 951–972 (2014). https://doi.org/10.1002/spe.2187
20. Zingg, S., Krstić, S., Raszyk, M., Schneider, J., Traytel, D.: Verified first-order monitoring with recursive rules. In: TACAS 2022. LNCS, vol. 13244, pp. 236–253. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-99527-0_13