


Towards Static Performance Guarantees for Programs with Run-Time Checks

Maximiliano Klemen

IMDEA Software Institute and Universidad Politécnica de Madrid (UPM), Spain


maximiliano.klemen@imdea.org

 <https://orcid.org/0000-0002-8503-8379>

Nataliia Stulova

IMDEA Software Institute and Universidad Politécnica de Madrid (UPM), Spain


nataliia.stulova@imdea.org

 <https://orcid.org/0000-0002-6804-2253>

Pedro Lopez-Garcia

IMDEA Software Institute and Spanish Council for Scientific Research (CSIC), Spain


pedro.lopez@imdea.org

 <https://orcid.org/0000-0002-1092-2071>

José F. Morales

IMDEA Software Institute, Spain


josef.morales@imdea.org

 <https://orcid.org/0000-0001-9782-8135>

Manuel V. Hermenegildo

IMDEA Software Institute and Universidad Politécnica de Madrid (UPM), Spain

manuel.hermenegildo@imdea.org

 <https://orcid.org/0000-0002-7583-323X>

Abstract

This document is an extended abstract of the Technical Report CLIP-1/2018.0.

2012 ACM Subject Classification Theory of computation → Program semantics, Theory of computation → Program analysis, Theory of computation → Pre- and post-conditions, Theory of computation → Invariants

Keywords and phrases Run-time Checks, Assertions, Abstract Interpretation, Resource Usage Analysis

Digital Object Identifier 10.4230/OASICS.ICLP.2018.10

Category Extended Abstract

Related Version Technical Report CLIP-1/2018.0 [2], <https://arxiv.org/abs/1804.02380>.

Funding Research partially funded by EU FP7 *ENTRA* agreement no 318337, Spanish MINECO TIN2015-67522-C3-1-R *TRACES* project, and Madrid M141047003 *N-GREENS* program.

Dynamic programming languages, such as Prolog, are a popular programming tool for many applications (e.g., web programming, prototyping, and scripting) due to their flexibility. The lack of inherent mechanisms for ensuring program data manipulation correctness (e.g., via full static typing or other forms of full static built-in verification) has sparked the evolution of flexible solutions, including assertion-based approaches in (constraint) logic languages, *soft-*



© Maximiliano Klemen, Nataliia Stulova, Pedro Lopez-Garcia, José F. Morales, and Manuel V. Hermenegildo;

licensed under Creative Commons License CC-BY

Technical Communications of the 34th International Conference on Logic Programming (ICLP 2018).

Editors: Alessandro Dal Palu', Paul Tarau, Neda Saeedloei, and Paul Fodor; Article No. 10; pp. 10:1–10:2

OpenAccess Series in Informatics



OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and *gradual*-typing in functional languages, and contract-based approaches in imperative languages. A trait that many of these approaches share is that some parts of the specifications may be the subject of *run-time checking* (e.g., those that cannot be discharged statically in systems that support this functionality). However, such run-time checking comes at the price of overhead during program execution, that can affect execution time, memory use, energy consumption, etc., often in a significant way.

Reducing run-time checking overhead is a challenging problem. Proposed approaches include discharging as many checks as possible via static analysis, optimizing the dynamic checks themselves, or limiting run-time checking points. Nevertheless, there are cases in which a number of checks cannot be optimized away and must remain in place, because of software architecture choices (e.g., the case of the external interfaces of reusable libraries or servers), the need to ensure a high level of safety (e.g., in safety-critical systems), etc.

At the same time, low program performance may not always be due to the run-time checks. A technique that can help in this context is *profiling*, often used to detect performance “hot spots” and guide program optimization. Prior work on using profiling in the context of optimizing the performance of programs with run-time checks clearly demonstrates the benefits of this approach. Still, profiling infers information that is valid only for some particular input data values, and thus the results obtained may not be valid for other inputs, and thus detecting the worst cases can take a long time, and is impossible in general.

We propose a method that uses *static cost analysis* (instead of – or as a complement to – dynamic profiling) to infer *upper and lower bounds* (guarantees) on the costs introduced by the run-time checks in a program (i.e., on the run-time checking overhead). Such bounds are safe, in the sense that are guaranteed to never be violated in actual executions. Since such costs are data-dependent, these bounds take the form of functions that depend on certain characteristics (generally, data sizes) of the program inputs. Our method provides the programmer with feedback and guarantees at compile-time regarding the overhead that run-time checking will introduce. Unlike profiling, the bounds provided hold for all possible execution traces, and allow assessing how such overhead varies with the size of the input. We also propose an assertion-based mechanism (as an extension to the Ciao assertion verification framework [1]) that allows programmers to specify bounds on the admissible overhead introduced by run-time checking. Our method then statically and automatically compares the inferred run-time checking overhead against the admissible levels and provides guarantees on whether the instrumented program conforms with the specifications.

We formalize and implement the method in the context of the Ciao assertion language and the CiaoPP verification framework, and present results from its experimental evaluation. Such results suggest that our method is feasible and also promising in providing bounds that help the programmer understand at the algorithmic level the overheads introduced by the run-time checking required for the assertions in the program, in different scenarios, such as performing full run-time checking or checking only the module interfaces.

References

- 1 M.V. Hermenegildo, F. Bueno, M. Carro, P. López, E. Mera, J.F. Morales, and G. Puebla. An Overview of Ciao and its Design Philosophy. *TPLP*, 12(1–2):219–252, 2012.
- 2 M. Klemen, N. Stulova, P. Lopez-Garcia, J. F. Morales, and M. V. Hermenegildo. An Approach to Static Performance Guarantees for Programs with Run-time Checks. Technical Report CLIP-1/2018.0, The CLIP Lab, IMDEA Software Institute and T.U. Madrid, April 2018. [arXiv:1804.02380](https://arxiv.org/abs/1804.02380).