On Optimally Combining Static and Dynamic Analyses For Intensional Program Properties

Ravi Mangal David Devecsery Alessandro Orso

The goal of algorithmic program verification is to automatically prove programs correct with respect to logical specifications. Program verification techniques can be broadly classified as either static or dynamic. Informally, static techniques try to establish the proof of correctness without executing the program. Program errors, if any, are detected prior to execution. However, there is no guarantee that a proof of correctness will be found, even when such a proof may actually exist. Dynamic techniques, on the other hand, insert run-time checks in the program that trigger an exception if the specification is violated during program execution.[1] Though these checks cause run-time overheads, they remove the burden of proving a program correct prior to execution.

Given the complementary nature of static and dynamic approaches to program verification, it is natural to consider a combined approach that first checks the program statically, and in case the correctness cannot be established, introduces run-time checks in the program wherever necessary. A number of such combined or hybrid approaches such as gradual typing [1,2], hybrid type checking [3], soft contract verification [4] have been proposed in the literature. Though these approaches differ in their details, the common thread is that they use static checking wherever possible, and only defer to run-time checks where necessary. These run-time checks are expressed using dynamic contracts [5,6].

Though dynamic contracts suffice for enforcing functional correctness properties, many program properties, such as data-race freedom, deadlock freedom, and information-flow security, need additional mechanisms for run-time enforcement. In particular, these properties require defining an instrumented language semantics that tracks extra information about how computations execute, beyond that which would appear in a standard semantics for the language. The extra information tracked is property-specific; for instance, data-race freedom requires keeping track of a may-happen-before ordering relation for reads and writes in the program. In general, such program properties that refer to "how computations execute" are dubbed as intensional properties. Dynamic analyses for checking intensional properties can have large overheads since, in addition to checking dynamic contracts, the analysis is also required to maintain and update additional program state. A hybrid approach can help reduce the run-time overheads, but the design of such hybrid approaches is more involved in the instrumented semantics setting. It does not suffice to reduce the number of dynamic contracts in the program. We also need to reduce the additional information tracked by the semantics. Fortunately, static analysis can help us ensure that only the necessary information is tracked in addition to ensuring that dynamic contracts are introduced only

[1]Note that, by "dynamic analysis", we do not mean approaches like testing, fuzzing, dynamic symbolic execution, or execution trace analysis which have also been referred to as dynamic analyses in the literature.

when necessary. A number of bespoke, ad-hoc combination of dynamic and static analyses for intensional properties already exist [7–10]. Such hybrid analyses, while effective, do not provide a general recipe or any general guarantees. The analysis designer has to repeat the hybrid analysis design process for every new pair of analyses.

In this talk, we present a precise formulation of hybrid analyses for intensional program properties. Moreover, we formally define the notion of an *optimally* efficient combination of static and dynamic analyses. For this purpose, we define the notion of *parametric* static and dynamic analyses, where the parameter controls the cost and precision of these analyses. We also propose a general recipe for constructing an optimal hybrid analysis given corresponding parametric static and dynamic analyses.

A key advantage of our formulation is that many existing static and dynamic analyses for intensional program properties can be easily expressed as parametric analyses, and combined to produce optimistic hybrid analyses using our recipe. This allows us to quickly construct the hybrid version of the analysis from the already existing static and dynamic analyses. In general, we view our formulation as a new point in the design space of general hybrid checking techniques.

References

[1] J. G. Siek and W. Taha, "Gradual typing for functional languages," in *In Scheme And Functional Programming Workshop*, 2006.

[2] J. G. Siek, M. M. Vitousek, M. Cimini, and J. T. Boyland, "Refined Criteria for Gradual Typing," in *1st Summit on Advances in Programming Languages (SNAPL 2015)*, 2015.

[3] K. Knowles and C. Flanagan, "Hybrid type checking," *ACM Trans. Program. Lang. Syst.*, 2010.

[4] P. C. Nguyen, S. Tobin-Hochstadt, and D. Van Horn, "Soft contract verification," in *Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming*, ICFP '14, 2014.

[5] B. Meyer, *Eiffel: The Language.* 1992.

[6] R. B. Findler and M. Felleisen, "Contracts for higher-order functions," in *Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming*, ICFP '02, 2002.

[7] W. Chang, B. Streiff, and C. Lin, "Efficient and extensible security enforcement using dynamic data flow analysis," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, 2008.

[8] J.-D. Choi, K. Lee, A. Loginov, R. O'Callahan, V. Sarkar, and M. Sridharan, "Efficient and precise datarace detection for multithreaded object-oriented programs," in *Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation*, PLDI '02, 2002.

[9] D. Lee, P. M. Chen, J. Flinn, and S. Narayanasamy, "Chimera: Hybrid program analysis for determinism," in *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '12, 2012.

[10] D. Devecsery, P. M. Chen, J. Flinn, and S. Narayanasamy, "Optimistic hybrid analysis: Accelerating dynamic analysis through predicated static analysis," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, 2018.