A Unified Lattice Model and Framework for Purity Analyses

Dominik Helm, Florian Kübler, Michael Eichberg, Michael Reif, Mira Mezini[1]

Abstract: This paper was presented in 2018 at *the 33rd ACM/IEEE International Conference on Automated Software Engineering* and proposes a framework for purity analyses. Analyzing methods in object-oriented programs whether they are side-effect free and also deterministic, i.e., *mathematically pure*, has been the target of extensive research. Identifying such methods helps to find code smells and security related issues, and helps analyses detecting concurrency bugs. *Pure* methods are further used for formal specifications and proving the *pureness* is necessary to ensure correct specifications. However, no common terminology exists which describes the purity of methods. Furthermore, some terms (e.g., *pure* or *side-effect free*) are used inconsistently. Further, all current approaches only report selected purity information making them only suitable for a smaller subset of the potential use cases. We present a fine-grained unified lattice model which puts the purity levels found in the literature into relation and which adds a new level that generalizes existing definitions. We have also implemented a scalable, modularized purity analysis which produces significantly more precise results for real-world programs than the best-performing related work. The analysis shows that all defined levels are found in real-world projects.

Keywords: Purity; Side-effects; Static Analysis; Lattice; Java.

1 Summary

Identifying side-effect free and also deterministic, i.e., *mathematically pure*, methods in object-oriented programs helps to improve subsequent analyses for finding bugs [Fi08]. Pure methods—written in programming languages such as Java—are also used by formal verification approaches as the foundation for the respective specifications [DL07]. In that case, it is necessary to prove a method's purity to ensure that the formal specifications are correct. Recent trends towards a more functional style of programming relying on pure methods, also demonstrate the overall relevance.

We present a fine-grained unified lattice model for specifying a method's purity. In the model, each of the 13 lattice elements has a well-defined semantics and is put into relation to the purity levels found in the literature. In addition to previously defined purity levels, the model is extended by the level *Contextual Purity* which generalizes the so-called *External Purity* [BF09] for methods that may modify their parameters but no static state. Being able to ignore specific operations in specific contexts [SCOD16], e.g., logging in business

[1] Technische Universität Darmstadt, FG Softwaretechnik, Germany
{helm,kuebler,eichberg,reif,mezini}@cs.tu-darmstadt.de

doi:10.18420/se2019-10

applications, is also supported and generalized to *Domain-specific Purity*. The proposed model is sufficiently detailed for all identified use cases. Furthermore, the purity levels (*External* and *Contextual* Purity) support purity analyses to rate methods as pure if the called methods have side-effects that are limited to the caller. Therefore, the lattice model is also a suitable target for modular purity analyses that reason about each method in isolation.

Additionally, we present a scalable, purity analysis, called OPIUM which is implemented using OPAL [Ei18], that produces more precise results for real-world code than the best-performing state-of-the-art tool ReIm [HM12]. OPIUM can, e.g., compute the purity for all ≈16000 methods of Batik—which requires computation of the purity of all library methods transitively used by Batik—in 103s on a modern computer which is on par with Batik while it analyzes Xalan in 104s, ≈25% faster than ReIm. OPIUM not only identifies more *side-effect free* methods that state-of-the-art tools including ReIm, but it also provides more fine-grained results by reporting *pure*, *externally* and *contextually pure* and *side-effect free* methods as well as *domain-specific* methods that adhere to these properties only in a given domain, e.g. excluding logging or exceptions. Using the analysis, we show that all defined levels are relevant when analyzing real-world projects such as those from XCorpus or the Java and Scala Development Kits. The analysis infers purity for individual methods in isolation. It relies on the results of several independent analyses, making it modularly composable with supporting analyses with different precision/performance trade-offs.

OPIUM is available at `www.opal-project.de/Opium.html`.

References

[BF09] Benton, William C; Fischer, Charles N: Mostly-functional behavior in Java programs. In: International Workshop on Verification, Model Checking, and Abstract Interpretation. Springer, pp. 29–43, 2009.

[DL07] Darvas, Ádám; Leino, K Rustan M: Practical reasoning about invocations and implementations of pure methods. In: FASE. volume 4422. Springer, pp. 336–351, 2007.

[Ei18] Eichberg, Michael; Kübler, F; Helm, D; Reif, M; Salvaneschi, G; Mezini, M: Lattice Based Modularization of Static Analyses. In: Companion Proceedings for the ISSTA/ECOOP 2018 Workshopss. ACM, pp. 111–116, 2018.

[Fi08] Finifter, Matthew; Mettler, Adrian; Sastry, Naveen; Wagner, David: Verifiable functional purity in Java. In: Proceedings of the 15th ACM conference on Computer and communications security. ACM, pp. 161–174, 2008.

[HM12] Huang, Wei; Milanova, Ana: ReImInfer: Method purity inference for Java. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. ACM, p. 38, 2012.

[SCOD16] Stewart, Arran; Cardell-Oliver, Rachel; Davies, Rowan: Fine-grained classification of side-effect free methods in real-world Java code and applications to software security. In: Proceedings of the Australasian Computer Science Week Multiconference. ACM, p. 37, 2016.