

# Verified Abstractions on Developers' Desks (Invited Talk)

Viktor Kuncak

<http://lara.epfl.ch/~kuncak>

**Verified data abstraction.** Data abstraction has long been recognized as a methodology to decompose the verification problem into data structure implementation and the algorithm that uses the data structure [3, 8]. Today we are close to being in a position to incorporate verified data abstraction into programming languages and systems. We discuss the ingredients needed to make verified data abstraction practical: program verification, specification-based testing, program analysis, programming methodology, as well as programming language design. We draw our examples from the systems Hob [5] and Jahob [4, 12] that implement verified data abstractions, as well as our recent work on verifying properties of Scala programs [2].

**Automated reasoning.** Decision procedures and their combination [11] are the basis of automated verification. Of particular importance for programs with references are decision procedures for reasoning about reachability [11] and about collections of objects [6, 9]. Developer-friendly annotation languages for proving complex properties [12] allow library designers to develop and verify efficient data structures whose correctness may rely on complex invariants.

**Program analysis.** Program analyses can infer loop invariants within data structures [10] but can be used to check data structure usage protocols [7]. Checking usage protocols is important because it provides a concrete benefit of verified abstractions to data structure clients. In addition to type-state analysis, we describe a client analysis for a language with user-defined “deconstruction” (pattern matching) [2]. For such analysis, it is more natural to use algebraic style instead of model-based specifications.

**Abstraction of mutable state.** For the case of mutable data structures, we classify the approaches into those that *admit* performing internal modifications, and those that *pretend not to* do such modifications. Approaches that admit modification can make the verification of clients difficult because of proof obligations on the invisible state. Approaches that pretend not to do modifications often come with limitations on how the implementation can be written, and sometimes depend on one-directional nature of references.

**Language design.** Among the simple language mechanisms that can eliminate some of the complex encapsulation reasoning we discuss the role of object-external relations [7], and a more aggressive use of value types. Going back to functional programming and systems such as SETL [1], we claim that programming systems should

have a substantial collection of built-in core abstractions such as sets, relations, maps, multisets, and lists. This will provide users with a vocabulary for program specification, and enable the programming system to deploy specialized automated reasoning procedures to perform sophisticated verification and compilation tasks.

**Pluggable methodologies.** To enable easy client reasoning about complex user-defined abstractions without subscribing one particular verification ideology, we suggest a verification system that has explicit representation of the operational semantics of the language and allows proofs of meta-theoretic lemmas. Among the uses of such system are multiple pluggable confinement methodologies within one piece of software.

## References

- [1] R. K. Dewar. Programming by refinement, as exemplified by the SETL representation sublanguage. *ACM TOPLAS*, July 1979.
- [2] M. Dotta, P. Suter, and V. Kuncak. On static analysis for expressive pattern matching. Technical Report LARA-REPORT-2008-004, EPFL, 2008.
- [3] C. A. R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1(4):271–281, 1971.
- [4] V. Kuncak. *Modular Data Structure Verification*. PhD thesis, EECS Department, Massachusetts Institute of Technology, February 2007.
- [5] V. Kuncak, P. Lam, K. Zee, and M. Rinard. Modular pluggable analyses for data structure consistency. *IEEE Transactions on Software Engineering*, 32(12), December 2006.
- [6] V. Kuncak and M. Rinard. Towards efficient satisfiability checking for Boolean Algebra with Presburger Arithmetic. In *CADE-21*, 2007.
- [7] P. Lam, V. Kuncak, and M. Rinard. Generalized typestate checking for data structure consistency. In *6th Int. Conf. Verification, Model Checking and Abstract Interpretation*, 2005.
- [8] R. Milner. An algebraic definition of simulation between programs. In *Proceedings of the 2nd international joint Artificial intelligence Conference*, 1971.
- [9] R. Piskac and V. Kuncak. Linear arithmetic with stars. In *CAV*, 2008.
- [10] T. Wies, V. Kuncak, K. Zee, A. Podelski, and M. Rinard. Verifying complex properties using symbolic shape analysis. In *Workshop on Heap Abstraction and Verification (collocated with ETAPS)*, 2007.
- [11] T. Wies, R. Piskac, and V. Kuncak. Combining theories with shared set operations. In *FroCoS: Frontiers in Combining Systems*, 2009.
- [12] K. Zee, V. Kuncak, and M. Rinard. Full functional verification of linked data structures. In *ACM Conf. Programming Language Design and Implementation (PLDI)*, 2008.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

IWACO '09, July 6 2009, Genova, Italy

Copyright © 2009 ACM 978-1-60558-546-8/09/07...\$5.00.