Verified Analysis of Functional Data Structures*

Tobias Nipkow

Technische Universität München, Munich, Germany

Abstract

In recent work the author has analyzed a number of classical functional search tree and priority queue implementations with the help of the theorem prover Isabelle/HOL. The functional correctness proofs of AVL trees, red-black trees, 2-3 trees, 2-3-4 trees, 1-2 brother trees, AA trees and splay trees could be automated. The amortized logarithmic complexity of skew heaps, splay trees, splay heaps and pairing heaps had to be proved manually.

1998 ACM Subject Classification F.3.1 Specifying and Verifying Programs and F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Program Verification, Algorithm Analysis, Functional Programming

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.4

Category Invited Talk

1 Summary

Recent work on the analysis of functional data structures [6, 7] considers two questions: functional correctness and amortized complexity. In the theorem proving community, functional correctness of programs is the primary issue and their complexity is analyzed much less frequently. In the algorithms community it is the other way around: functional correctness is often viewed as obvious and the main issue is the complexity. We confirm the latter point of view in two case studies involving a number of functional search tree and priority queue implementations. The proofs were all conducted with the help of the theorem prover Isabelle/HOL [8, 9].

In [7] it is shown how to automate the functional correctness proofs of insertion and deletion in search trees: by means of an inorder traversal function that projects trees to lists, the proofs are reduced from trees to lists. With the help of a small lemma library, functional correctness and preservation of the search tree property are proved automatically for a range of data structures: unbalanced binary trees, AVL trees, red-black trees, 2-3 trees, 2-3-4 trees, 1-2 brother trees, AA trees and splay trees.

In [6] a framework for the analysis of the amortized complexity of (functional) data structures is formalized and applied to a number of standard examples and to three famous non-trivial ones: skew heaps, splay trees and splay heaps. More recently, pairing heaps were added in collaboration with Hauke Brinkop [2, 5]. In all cases we proved logarithmic amortized complexity and the proofs were largely manual, following the existing algorithms literature.

* Supported by DFG grant NI 491/16-1.

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 4; pp. 4:1–4:2
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2 Related Work

Very close to the above work is Charguéraud' and Pottier's verification of the almost-linear amortized complexity of an OCaml implementation of Union-Find in Coq [3]. Using different methods but also aiming for performance analysis is work on automatic analysis of worst case execution time [11], analysis of complexity of term rewriting systems (e.g. [1, 10]), and automatic complexity analysis of functional programs (e.g. [4]).

References

1. Martin Avanzini, Georg Moser, and Michael Schaper. TcT: Tyrolean Complexity Tool. In M. Chechik and J.-F. Raskin, editors, *TACAS*, volume 9636 of *LNCS*, pages 407–423. Springer, 2016.
2. Hauke Brinkop. Verifikation der amortisierten Laufzeit von Pairing Heaps in Isabelle. Bachelor's thesis, Fakultät für Informatik, Technische Universität München, 2015.
3. Arthur Charguéraud and François Pottier. Machine-checked verification of the correctness and amortized complexity of an efficient union-find implementation. In C. Urban and X. Zhang, editors, *ITP 2015*, volume 9236 of *LNCS*, pages 137–153. Springer, 2015.
4. Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. Multivariate amortized resource analysis. *ACM Trans. Program. Lang. Syst.*, 34(3):14, 2012.
5. Tobias Nipkow. Amortized complexity verified. *Archive of Formal Proofs*, 2014. `http://isa-afp.org/entries/Amortized_Complexity.shtml`, Formal proof development.
6. Tobias Nipkow. Amortized complexity verified. In C. Urban and X. Zhang, editors, *ITP 2015*, volume 9236 of *LNCS*, pages 310–324. Springer, 2015.
7. Tobias Nipkow. Automatic functional correctness proofs for functional search trees. In J. Blanchette and S. Merz, editors, *ITP 2016*, LNCS. Springer, 2015. To appear.
8. Tobias Nipkow and Gerwin Klein. *Concrete Semantics with Isabelle/HOL*. Springer, 2014. URL: `http://concrete-semantics.org`.
9. Tobias Nipkow, Lawrence Paulson, and Markus Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
10. Lars Noschinski, Fabian Emmes, and Jürgen Giesl. Analyzing innermost runtime complexity of term rewriting by dependency pairs. *J. Autom. Reasoning*, 51(1):27–56, 2013.
11. Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David B. Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter P. Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem – overview of methods and survey of tools. *ACM Trans. Embedded Comput. Syst.*, 7(3), 2008.