Automated Reasoning and Natural Proofs for Programs Manipulating Data Structures

P. Madhusudan

University of Illinois at Urbana-Champaign, USA
and visiting Microsoft Research, Bangalore, INDIA
madhu@illinois.edu

Abstract

We consider the problem of automatically verifying programs that manipulate a dynamic heap, maintaining complex and multiple data-structures, given modular pre-post conditions and loop invariants. We discuss specification logics for heaps, and discuss two classes of automatic procedures for reasoning with these logics. The first identifies fragments of logics that admit completely decidable reasoning. The second is a new approach called the *natural proof method* that builds proof procedures for very expressive logics that are automatic and sound (but incomplete), and that embody natural proof tactics learnt from manual verification.

1998 ACM Subject Classification D.2.4 Software/Program Verification, F.3.1 Specifying and Verifying and Reasoning about Programs, F.4.1 Mathematical Logic

Keywords and phrases logic, heap structures, data structures, program verification

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.34

Summary

One of the most promising paradigms of software verification is *automated deductive verification*, which combines user written modular annotations for specifications as well as invariants (pre/post conditions, loop invariants, assertions, ghost code, etc.) and automatic theorem proving of the resulting verification conditions. This paradigm is extremely powerful as it appears to be a rich enough paradigm using which any reliable software can be built (unlike completely automatic approaches) and because the user needs to specify only modular and loop annotations, leaving reasoning entirely to automatic techniques. Several success stories of large software verification projects attest to the power of this paradigm (the Verve OS project [8], the Microsoft hypervisor verification project using VCC [1], and a recent verified-for-security OS+browser for mobile applications [5], to name a few).

Verification conditions do not, however, always fall into decidable theories. In particular, the verification of properties of the *dynamically modified heap* is a big challenge for logical methods. The dynamically manipulated heap poses several challenges, as typical correctness properties of heaps require complex combinations of structure (e.g., p points to a tree structure, or to a doubly-linked list, or to an almost balanced tree, with respect to certain pointer-fields), data (the integers stored in data-fields of the tree respect the binary search tree property, or the data stored in a tree is a max-heap), and separation (the procedure modifies one list and not the other and leaves the two lists disjoint at exit, etc.). The fact that the dynamic heap contains an unbounded number of locations means that expressing the above properties requires *quantification* in some form, which immediately precludes the use of most decidable theories currently handled by SMT solvers.

We will discuss two logics that have emerged in this regime— classical logic augmented with ghost-code and separation logic [6]. We will then discuss two thrusts in automatically verifying the resulting verification conditions— decidable logics and natural proofs.

32nd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012).
Editors: D. D'Souza, J. Radhakrishnan, and K. Telikepalli; pp. 34–35

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Decidable logics: We discuss several decidable logics that restrict classical logics with quantification so that they are amenable to automated reasoning. In particular, we will discuss the class of STRAND logics [2, 3] that admit decision procedures for data-structures combining tree-interpretable structure and data, with restriction of relations to be *elastic*. We will explore the boundary between decidability and undecidability in this domain.

Decidable logics that are expressive enough to handle common data-structures and their correctness quickly become awkward, and perhaps more importantly, their decision procedures get incredibly complex, failing to mimic the simple proofs that are sufficient to prove many correct programs correct. This leads us to the quest for simple proofs.

Natural proofs: Natural proofs [4, 7] aim at discovering *simple* proofs— proofs that mimic human reasoning of programs, which often rely on *induction* on the shape of the data-structure, and combine unfolding recursive definitions on data-structures followed by unification and simple quantifier-free reasoning over specific theories. Natural proofs exploit a *fixed* set of proof tactics, keeping the expressiveness of powerful logics, retaining the automated nature of proving validity, but giving up on completeness (giving up decidability, retaining soundness).

We discuss the logic DRYAD, a dialect of separation logic, with no explicit (classical) quantification but with recursive definitions to express second-order properties. We show that DRYAD is both powerful in terms of expressiveness, and closed under the strongest-post with respect to bounded code segments. We also show that DRYAD can be systematically converted to classical logic using the theory of sets, and develop a natural proof mechanism for classical logics with recursion and sets that implements a sound but incomplete reduction to decidable theories that can be handled by an SMT solver.

We show, using a large class of correct programs manipulating lists, trees, cyclic lists, and doubly linked lists as well as multiple data-structures of these kinds, that the natural proof mechanism often succeeds in proving programs automatically. These programs are drawn from a range of sources, from textbook data-structure routines (binary search trees, red-black trees, etc.) to routines from Glib low-level C-routines used in GTK+/Gnome to routines implementing file-systems, a routine from the Schorr-Waite garbage collection algorithm, to several programs from a recent secure framework developed for mobile applications [5].

References

1 Ernie Cohen, Markus Dahlweid, Mark A. Hillebrand, Dirk Leinenbach, Michal Moskal, Thomas Santen, Wolfram Schulte, and Stephan Tobies. VCC: A practical system for verifying concurrent C. In *TPHOLs'09*, volume 5674 of *LNCS*, pages 23–42. Springer, 2009.

2 P. Madhusudan, Gennaro Parlato, and Xiaokang Qiu. Decidable logics combining heap structures and data. In *POPL'11*, pages 611–622. ACM, 2011.

3 P. Madhusudan and Xiaokang Qiu. Efficient decision procedures for heaps using STRAND. In *SAS'11*, volume 6887 of *LNCS*, pages 43–59. Springer, 2011.

4 P. Madhusudan, Xiaokang Qiu, and Andrei Stefanescu. Recursive proofs for inductive tree data-structures. In *POPL'12*, pages 123–136. ACM, 2012.

5 Haohui Mai, Edgar Pek, Hui Xue, P. Madhusudan, and Samuel King. Building a secure foundation for mobile apps. In *ASPLOS'13*. to appear, 2013.

6 Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In *CSL'01*, volume 2142 of *LNCS*, pages 1–19. Springer, 2001.

7 Xiaokang Qiu, Pranav Garg, Andrei Stefanescu, and P. Madhusudan. Natural proofs for structure, data, and separation. Unpublished manuscript, 2012.

8 Jean Yang and Chris Hawblitzel. Safe to the last instruction: automated verification of a type-safe operating system. In *PLDI'10*, pages 99–110. ACM, 2010.