How Efficient Can Fully Verified Functional Programs be - a Case Study of Graph Traversal Algorithms

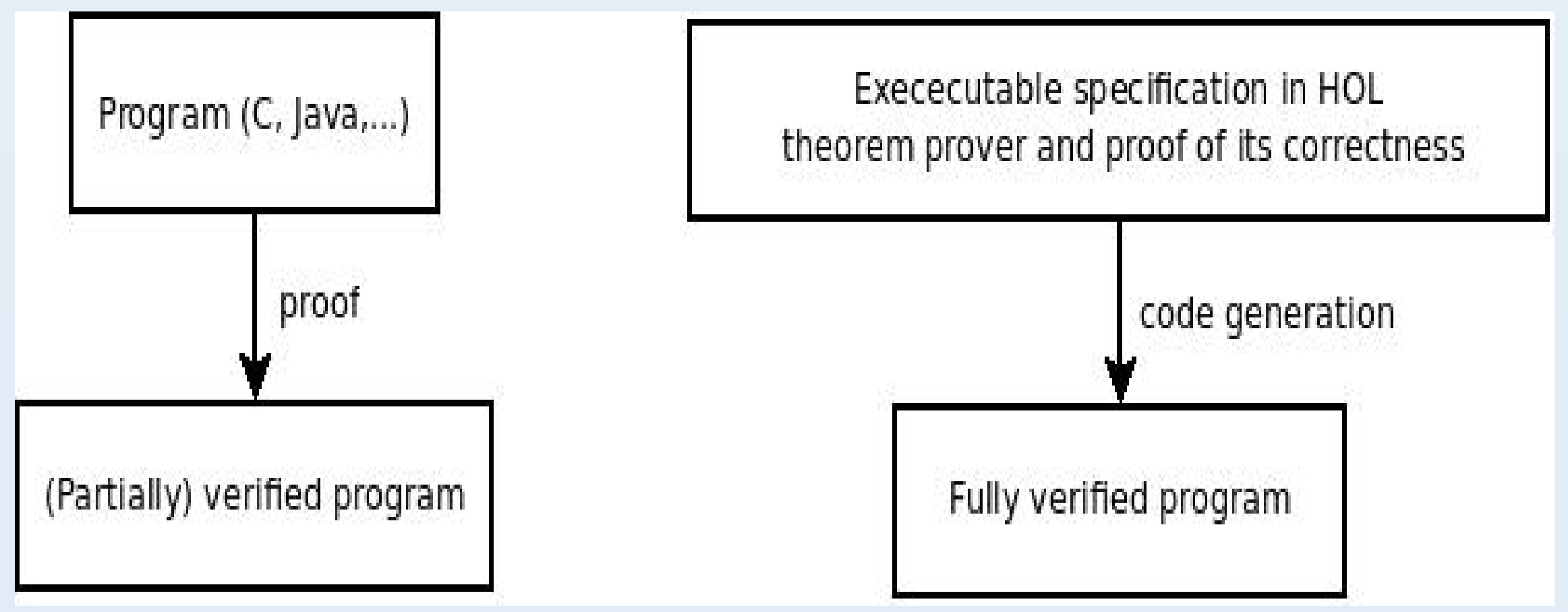
Mirko Stojadinović   mirkos@matf.bg.ac.rs
University of Belgrade, Faculty of Mathematics

Abstract

One approach in achieving fully verified software is formalizing the software within a proof assistant, proving its total correctness, and exporting executable code in a functional programming language by means of code generation. In order to be applicable for the real world applications, the generated code must be efficient. In this work we evaluate how efficient can verified programs be, by doing a case study of graph algorithms. We focus on BFS (Breadth first search) and specify it in Isabelle/HOL proof assistant in several ways: a purely functional recursive specification, and an imperative specification within the Imperative/HOL framework. The exported SML programs are compared to the unverified version implemented in C, and, although an order of magnitude slower, the Imperative/HOL version can still successfully handle very large graphs.

Background

Formal verification uses formal mathematical methods to prove correctness of programs. There are two main approaches to formal verification.


Program (C, Java,...)
proof
(Partially) verified program
Exececutable specification in HOL theorem prover and proof of its correctness
code generation
Fully verified program

Objectives

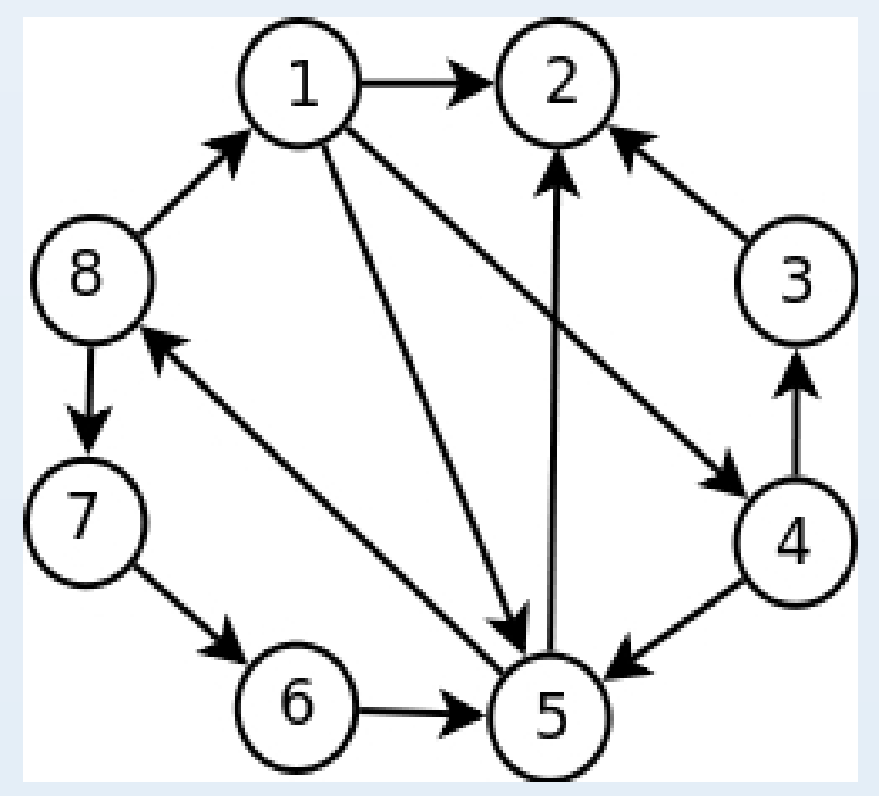
The main motivation for this work is to see if efficiency of fully verified functional programs (potentially using imperative data structures) can be comparable to efficiency of unverified imperative programs.

We focus on various graph algorithms and here we present BFS algorithm.

BFS algorithm

BFS algorithm traverses graph level by level. The description of algorithm is:

- for all vertices on current level (one vertex at the beginning) that have not been visited yet, mark them as visited and calculate the set of their neighbours
- repeat upper step with neighbours as vertices on a new level until there are unvisited neighbours.
- Example:



Traversal:

1

2 4 5

3 8

7

6

Isabelle/HOL is a generic proof assistant. It allows mathematical statements to be expressed in a formal language and provides tools for proving them in a logical calculus.

Implementation

4 versions:

1) Isabelle/HOL – using sets (fully verified)

2) Isabelle/HOL – using lists

3) Isabelle/HOL – using arrays (proving of its correctness is in progress)

4) C (unverified, used for efficiency comparison)

Results

- From the specifications of the first three versions, codes are exported to SML and their execution times are compared to execution time of the version implemented in C.
- Only the time needed for execution of algorithms is computed (e.g. time for input parsing and printing the results is excluded)
- Machine: PC Pentium (R) Dual-Core E6500 2.93GHz with 2GB RAM, running under Linux.

Vertices	Edges	SML (sets)	SML (lists)	SML (arrays)	C
64	640	0.8s	8.2s	0.003s	0.00001s
193	1930	4.3s	320s	0.003s	0.00001s
70 570	705 700	*	*	0.7s	0.02s
99 888	499 440	*	*	0.15s	0.03s
4 500	10 143 032	*	*	0.5s	0.05s
5 555	15 450 463	*	*	0.7s	0.07s

- First two versions are really inefficient
- C program is in average 10-20 times faster then SML program using arrays

Conclusion

- Graphs with thousands of vertices and millions of edges traversed within a second in SML using arrays
- SML program is slower then C program (expected, because e.g., verified program is using big numbers instead of integers to ensure overflows will not happen)
- We hope that good efficiency can be also obtained for some other graph algorithms
- Hopefully, verified functional implementations will show even better results if algorithms require more computations and less data-structure manipulation (contrary to BFS).

Future work

We plan to compare different approaches to verification by implementing and proving correctness of graph algorithms in other systems, e.g. Coq, Microsoft Boogie.

References

1. Nipkow T., Paulson C. L., Wenzel M. Isabelle/HOL: A Proof Assistant for Higher- Order Logic, volume 2283 of Lecture Notes. In Computer Science. Springer-Verlag, 2002.

2. Bulwahn, L., Krauss, A., Haftmann, F., Erkok, L., Matthews, J. Imperative Functional Programming with Isabelle/HOL. In: TPHOLs 08

3. Harrison J., Formal proof theory and practice. In: Notices of the American Mathematical Society, 55(11):1395-1406, December 2008.

4. Cormen T., Lesierson C., Rivest L., Stein C. Introduction to Algorithms. MIT Press, Cambridge, MA, 2nd edition, 2001.

5. Woodcock J. C. P., Davies J. Using Z: Specification, Refinement, and Proof. Prentice-Hall, 1996.